Contents

11 ADAPTIVE FILTERING **1**
11.1 THE WIENER FILTER REVISITED 6
11.2 STEEPEST DESCENT METHOD 7
11.2.1 Method of Steepest Descent 7
11.2.2 Performance Analysis 10
11.3 THE LMS ALGORITHM 18
11.3.1 A Simple Adaptive Algorithm 18
11.3.2 Discussion of LMS Convergence 20
11.3.3 Related Forms 26
11.4 LMS FOR LATTICES 32
11.4.1 Adaptive Linear Prediction 33
11.4.2 Joint Process Estimation 36
11.5 THE RLS ALGORITHM 39
11.5.1 Weighted Least Squares 39
11.5.2 Recursive Least Squares 42
11.5.3 RLS Performance 50

Chapter 11

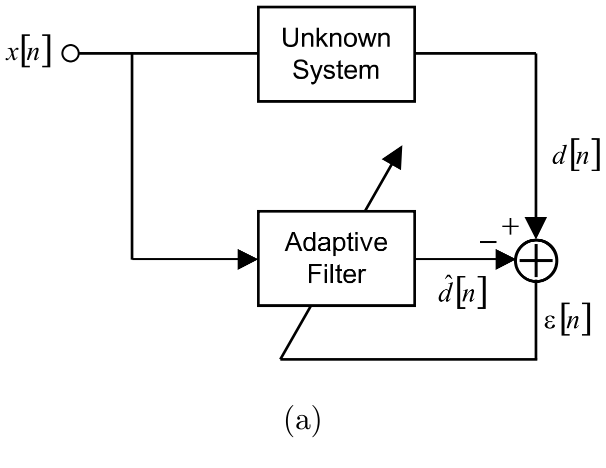
ADAPTIVE FILTERING

Chapter 7 deals with the problem of optimal filtering in its various forms. In all of these problems there is a desired signal $d[n]$ which is not directly observable, and a set of *observations* in the form of a signal $x[n]$ related to $d[n]$ (see Fig. 7.6). Since in that chapter there is an assumed *known* statistical relationship between $d[n]$ and $x[n]$, and the statistical properties of $x[n]$ itself are known, it is possible to develop a filter that estimates $d[n]$ from the observations to minimize the mean-square error.

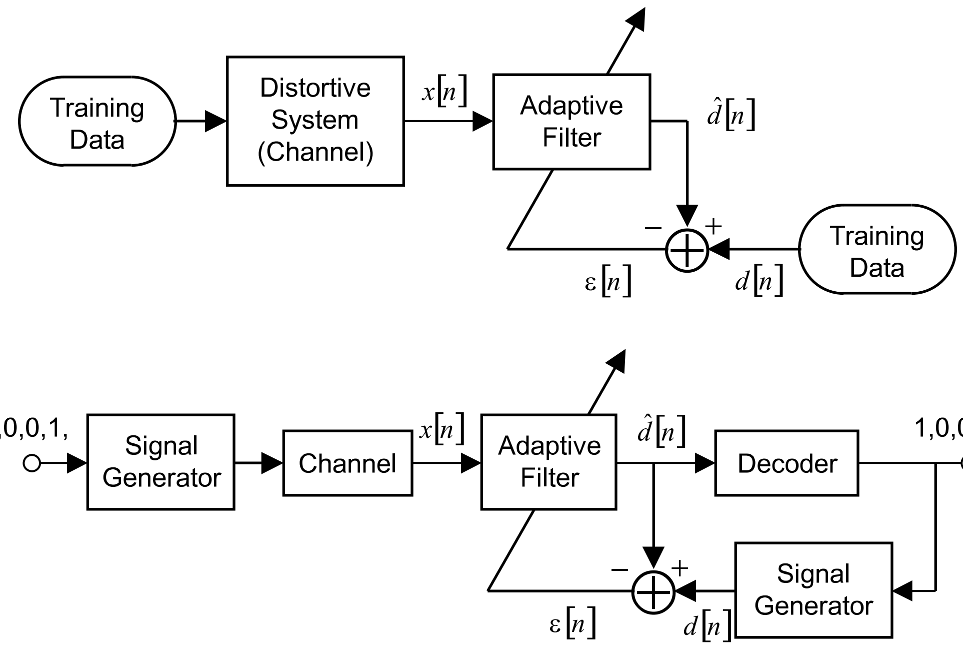
In many problems of interest the statistical parameters needed to develop the optimal filter are not known or may be changing with time in an unprescribed way. Consider, for example, data communication over an ordinary switched telephone line. There is dispersion and noise introduced on the line which prevent communication over an unconditioned line at rates higher than a few hundred bits per second. It is not possible to design a fixed filter to undo the distortion because the line characteristics are *different* every time one dials up, and may even change during the connection period. In cases such as this there is need for an adaptive filter.

While the adaptive filter does not require any *a priori* knowlege of the signal statistics, it does assume that the desired signal is somehow more accessible than would be implied in Fig 7.6. There are several general types of problems however for which this is the case. The four most common types of problems are illustrated in Fig. 11.1.

The first configuration corresponds to the system identification problem (Fig 11.1(a)). An unknown system and an adaptive filter, which serves as a model for that system, are provided with the same input. The output of the unknown system, which is available directly, represents the "desired signal"

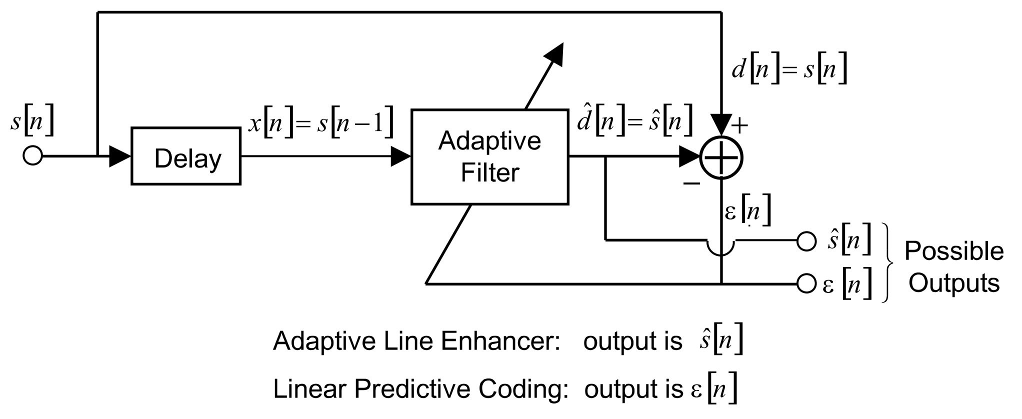


(a)



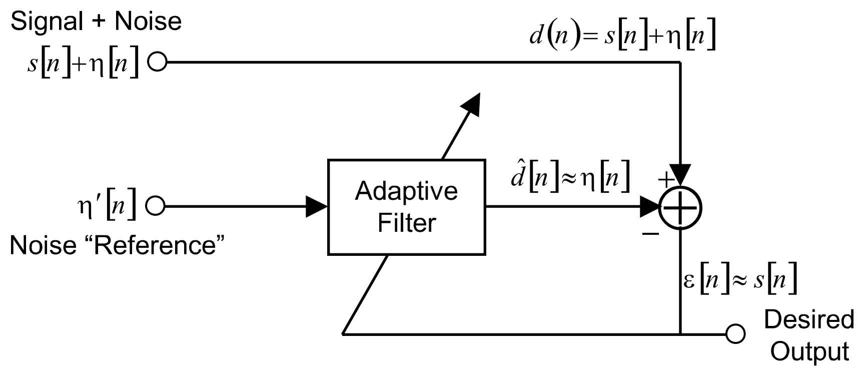
(b)

Figure 11.1: Common adaptive filter configurations. (a) System identification. (b) System equalization.



$s[n]$
Delay
$x[n] = s[n-1]$
Adaptive Filter
$\hat{d}[n] = \hat{s}[n]$
$d[n] = s[n]$
+
−
$\varepsilon[n]$
$\hat{s}[n]$
$\varepsilon[n]$
Possible Outputs
Adaptive Line Enhancer: output is $\hat{s}[n]$
Linear Predictive Coding: output is $\varepsilon[n]$

(c)



(d)

Figure 11.1: Common adaptive filter configurations, cont'd. (c) Linear prediction. (d) Noise (interference) cancellation.

$d[n]$ and the parameters of the adaptive filter are adjusted to minimize the mean-square error. Note how the error $\varepsilon[n]$ is "fed back" in the diagram, indicating that in all forms of adaptive filters the error sequence is used in the algorithm to adjust the parameters. The filter and its parameters after convergence of the adaptive algorithm represent a model for the unknown system.

The second configuration, shown in Fig. 11.1(b), represents the system or channel equalization problem that was introduced above. Here the adaptive filter approximates the inverse of the distortive system to mitigate its effects on the input data. In the upper part of the figure the adaptive filter is "set up" (initially adjusted) by sending a sequence of data (usually a pseudonoise sequence) that is known to both the sender and receiver. Since the sequence $d[n]$ which is being transmitted is known at the receiver end, it can be generated without error in order to "train" the filter. Therefore, the known data sequence is sometimes referred to as "training data." Once the filter has been adjusted to achieve some desired minimum error on the training data, the system is switched into a communication mode (if the application is data communication). The system may then operate in what is called a "decision-directed" mode shown in the lower part of the figure. The output of the adaptive filter is used to decode the data. For example, in a binary communication system, it may be simply passed through a threshold device to decide if the signal represents a one or a zero. A clean signal, based on this decision, can then be generated, fed back, and used as the desired signal $d[n]$ for the adaptive filter, which continues to adapt. The system operates in this decision-directed mode as long as not too many errors are being made. If a large number of errors starts to appear in the system (these can be detected for example by error-correcting codes) then the system may need to retrain and/or reduce the transmission rate.

The third common form of adaptive filter is the linear predictor, shown in Fig. 11.1(c). Here the desired signal is the input signal one or more steps ahead in time. It has been seen that linear prediction is a fundamental problem in signal processing that underlies a host of important applications. Two common uses for the linear predictor configuration in adaptive filtering are the adaptive line enhancer (ALE) [1] and the linear predictive coder. In the adaptive line enhancer, the input $s[n]$ consists of a narrowband information signal in additive broadband noise. Since the filter is better able to predict the narrowband signal than the noise, the resulting output is an enhanced version of the signal. In linear predictive coding, it is the error sequence rather than the signal itself which is transmitted. When the system has to perform over a large class of input signals with varying statistics, an adaptive filter can generally achieve a lower error than a fixed filter; consequently the coding can be more efficient.

The last common configuration that will be discussed is the noise or interference canceller shown in Fig. 11.1(d). This system has two inputs. The first consists of a signal $s[n]$ plus additive noise $\eta[n]$. The second is another noise sequence $\eta'[n]$ *highly correlated* with the first noise sequence but containing *none* (or very little) of the signal $s[n]$. The adaptive system is configured to estimate the contaminating noise source $\eta[n]$ from what is called the *reference* noise source $\eta'[n]$ and subtract it to provide a cleaner form of the signal $s[n]$. Note that in this configuration what is generically called the "desired signal" $d[n] = s[n] + \eta[n]$ is *not the desired output!* The ouput is actually the error signal $\varepsilon[n]$, which hopefully is close to the signal $s[n]$ after the noise is subtracted. An example of a practical system using this configuration is the so-called "active" headphones manufactured by the Bose Corporation and others for use in aircraft cockpits and other noisy environments. The acoustic signal arriving at the user's ear is the sum of the signal from the radio or other electronic source plus the noise from the cockpit. The headphones have ample opportunity to sample the cockpit noise through a tiny external microphone that does not sense any of the radio signal. This noise is processed by the adaptive filter and combined out of phase with the radio signal in the headphones so that the total signal arriving at the user's ear has reduced (cancelled) noise.

This chapter focuses on FIR adaptive filters, which form the larger part of all current practical applications. The convergence and computational properties of these FIR filters are fairly well understood. Adaptive IIR and even nonlinear filters are currently a field of research, but their study is well grounded in the theory presented here. An introduction to these topics can be found in a variety of references including [2, 3, 4, 5, 6].

The chapter begins with a review of the FIR Wiener filter, to establish some new notation and examine its properties from the point of view provided in optimization theory. The method of steepest descent is then applied to this problem and its properties are examined in an idealistic setting where the processes are wide sense stationary and all of the statistical parameters are known. This gives motivation and insight for a first adaptive method, the LMS algorithm, which although apparently simple, is perhaps the most widely used method in real applications because of its low computational complexity. Several variations on LMS are discussed, including lattice forms based on the same principles.

The chapter moves on to discuss adaptive least squares algorithms, which are related to the "data-oriented" least squares methods that are the subject of Chapter 9 in this book. As with their non-adaptive counterparts, the adaptive least squares methods do not rely upon any statistical ideas *per se* and so form a different class of algorithms. These algorithms, in fact, achieve an *exact* solution to a least squares formulation of the adaptive filtering problem, but do so at the expense of increased computation or decreased numerical stability. We begin with the standard Recursive Least Squares (RLS) algorithm and proceed to discussion of lattice implementations and finally the rather complicated but computationally efficient "fast" RLS algorithms.

11.1 THE WIENER FILTER REVISITED

Let us begin by reviewing the FIR Wiener filter for a stationary random process as discussed in Chapter 7. The filter and related signals are depicted in Fig. 11.2. The input $x[n]$ is processed through the filter to produce an

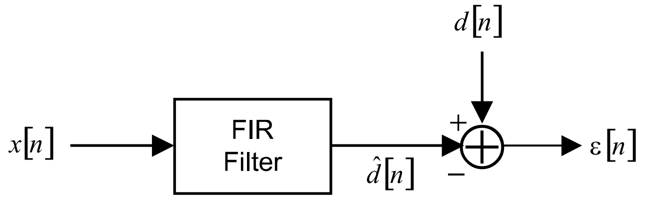


Figure 11.2: Optimal filtering problem.

estimate $\hat{d}[n]$ of the *desired* signal $d[n]$. The error ε is formed as the difference of d and $\hat{d}$.

Now let the filter be described by a set of weights $\{w_k\}$ so that the impulse response of the filter is given by

$$h[n] = \begin{cases} w_n & 0 \leq n \leq P-1 \\ 0 & \text{otherwise} \end{cases} \tag{11.1}$$

The filtering equation can then be written as

$$\hat{d}[n] = \sum_{k=0}^{P-1} w_k x[n-k] = \mathbf{w}^T \tilde{\boldsymbol{x}}[n] \tag{11.2}$$

where $\mathbf{w}$ is the vector of weights defined by

$$\mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_{P-1} \end{bmatrix}^T \tag{11.3}$$

and $\boldsymbol{x}[n]$ is the vector of samples of the input

$$\boldsymbol{x}[n] = \begin{bmatrix} x[n-P+1] & x[n-P+2] & \cdots & x[n] \end{bmatrix}^T \tag{11.4}$$

and the $\tilde{}$ in (11.2) denotes the reversal of this vector.

The *error* sequence is defined as

$$\varepsilon[n] = d[n] - \hat{d}[n] = d[n] - \mathbf{w}^T \tilde{\boldsymbol{x}}[n] \tag{11.5}$$

The optimal (Wiener) filter is designed to minimize the mean-square error $\sigma_\varepsilon^2 = \mathcal{E}\{|\varepsilon[n]|^2\}$ and is determined by the Wiener-Hopf equation,

$$\boxed{\mathbf{R}_{\boldsymbol{x}} \mathbf{w}_\mathrm{o} = \tilde{\mathbf{r}}_{d\boldsymbol{x}}} \tag{11.6}$$

[see Chapter 7] where $\mathbf{R}_{\boldsymbol{x}}$ is the correlation matrix for the input process, and $\mathbf{r}_{d\boldsymbol{x}}$ is the cross-correlation vector between the desired signal and the input:

$$\mathbf{R}_{\boldsymbol{x}} = \mathcal{E}\{\boldsymbol{x}[n]\boldsymbol{x}^{*T}[n]\} \qquad \mathbf{r}_{d\boldsymbol{x}} = \mathcal{E}\{d[n]\boldsymbol{x}^*[n]\}$$

The symbol $\mathbf{w}_\mathrm{o}$ denotes the Wiener *optimal* weight vector that satisfies (11.6). The minimum mean-square error is then given by

$$\sigma_{\varepsilon_\mathrm{o}}^2 = R_d[0] - \mathbf{w}_\mathrm{o}^{*T} \tilde{\mathbf{r}}_{d\boldsymbol{x}} \tag{11.7}$$

where $R_d[0]$ is the correlation function for $d[n]$ evaluated at lag zero, and where we have denoted the error produced by the optimal filter as

$$\varepsilon_\mathrm{o}[n] = d[n] - \mathbf{w}_\mathrm{o}^T \tilde{\boldsymbol{x}}[n] \tag{11.8}$$

11.2 STEEPEST DESCENT METHOD

11.2.1 Method of Steepest Descent

In this section an iterative method is developed for solving the Wiener-Hopf equation. This will eventually lead to an algorithm for adaptive filtering.

Begin by writing a general equation for the mean-square error for an arbitrary set of filter weights $\mathbf{w}$:

$$\sigma_\varepsilon^2(\mathbf{w}) = \mathcal{E}\left\{|\varepsilon[n]|^2\right\} = \mathcal{E}\left\{(d[n] - \mathbf{w}^T\tilde{\boldsymbol{x}}[n])(d[n] - \mathbf{w}^T\tilde{\boldsymbol{x}}[n])^*\right\}$$

Expanding and taking the expectation then reduces this to the form[1]

$$\sigma_\varepsilon^2(\mathbf{w}) = R_d[0] - \mathbf{w}^{*T}\tilde{\mathbf{r}}_{d\boldsymbol{x}} - \tilde{\mathbf{r}}_{d\boldsymbol{x}}^{*T}\mathbf{w} + \mathbf{w}^{*T}\mathbf{R}_{\boldsymbol{x}}\mathbf{w} \tag{11.9}$$

This error surface, plotted as a function of the weights $\{w_k\}$, is a convex downward quadratic function; that is, in two dimensions it looks like a bowl (see Fig. 11.3). The minimum mean-square error $\sigma_{\varepsilon_\mathrm{o}}^2$ is *unique* and occurs

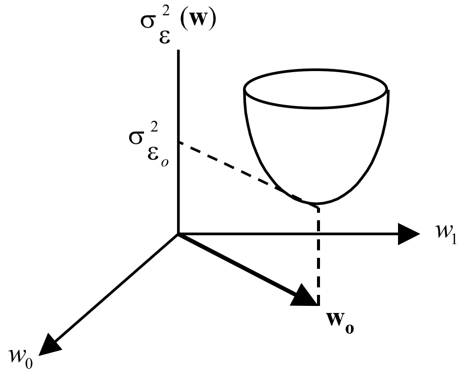

$\sigma_\varepsilon^2(\mathbf{w})$
$\sigma_{\varepsilon_o}^2$
w_1
$\mathbf{w_o}$
w_0

Figure 11.3: Error surface for the optimal filtering problem.

at the "bottom" of the surface, for which $\mathbf{w} = \mathbf{w}_\mathrm{o}$. [Note that when $\mathbf{w} = \mathbf{w}_\mathrm{o}$ the last two terms in this equation disappear due to (11.6), so that (11.9) reduces to (11.7).] Further, there are *no local minima.*

Now given that $\mathbf{w}$ is at some arbitrary value not equal to $\mathbf{w}_\mathrm{o}$, we would like to develop an algorithm so that $\mathbf{w}$ can be iteratively moved closer and eventually converge to the optimum value. Notice that when $\mathbf{w}$ is thus made a function of time, the mean-square error $\sigma_\varepsilon^2(\mathbf{w})$ and the error surface become functions of time. Nevertheless, the form of the error surface at any epoch in time is quadratic and convex downward.

The algorithm for iteratively changing the weights is known as the method of *steepest descent.* The weights are moved in the direction corresponding to

[1]The last term of this expansion, $\mathbf{w}^T\tilde{\mathbf{R}}_{\boldsymbol{x}}\mathbf{w}^*$ is inherently real-valued and can be replaced by the conjugated expression $\mathbf{w}^{*T}\tilde{\mathbf{R}}_{\boldsymbol{x}}^*\mathbf{w}$. Since the correlation matrix is Hermitian and Toeplitz, $\tilde{\mathbf{R}}_{\boldsymbol{x}}^* = \mathbf{R}_{\boldsymbol{x}}$ and (11.9) follows.

the negative gradient of the error surface. This is the steepest direction "downhill," and if the step size is not too large, this policy insures moving to the optimal bottom point of the surface (although not necessarily in the shortest time). Specifically, the weight vector is changed according to

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \mu \nabla_{\mathbf{w}^*} \sigma_\varepsilon^2 \tag{11.10}$$

where μ is a positive parameter representing the step size, and $\nabla_{\mathbf{w}^*} \sigma_\varepsilon^2$ represents the complex gradient of the error surface.[2]

Let us consider some explicit forms of (11.10) by evaluating the gradient. Applying the rules summarized in Table A.2 of Appendix A to (11.9) yields

$$\nabla_{\mathbf{w}^*} \sigma_\varepsilon^2 = -\tilde{\mathbf{r}}_{d\boldsymbol{x}} + \mathbf{R}_{\boldsymbol{x}} \mathbf{w}$$

[Note that setting this gradient equal to zero results in the Wiener-Hopf equation (11.6).] If this expression is substituted in (11.10) we obtain

$$\boxed{\mathbf{w}[n+1] = \mathbf{w}[n] + \mu(\tilde{\mathbf{r}}_{d\boldsymbol{x}} - \mathbf{R}_{\boldsymbol{x}} \mathbf{w}[n])} \tag{11.11}$$

This form represents a well-known algorithm for solving linear equations such as the Wiener-Hopf equation. The *equation error* $\tilde{\mathbf{r}}_{d\boldsymbol{x}} - \mathbf{R}_{\boldsymbol{x}} \mathbf{w}[n]$ is multiplied by a constant and added to the current value of the solution $\mathbf{w}[n]$ to obtain the solution $\mathbf{w}[n+1]$ for the next iteration. It was observed, that in our case the error surface is quadratic, so that the solution should indeed converge if μ is not too large. The specific details of convergence are discussed in Section 11.2.2.

[2]Note that if the problem involves real- rather than complex-valued signals, the algorithm should be written as

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \frac{\mu}{2} \nabla_{\mathbf{w}} \sigma_\varepsilon^2$$

where $\nabla_{\mathbf{w}} \sigma_\varepsilon^2$ is the real form of the gradient (see Appendix A). In this way all of the results that are derived from this point on follow in both the real and complex cases. In fact, if the weights w_k are written as $w_k = w_{Rk} + \jmath w_{Ik}$ then (11.10) is equivalent to the pair of equations

$$\begin{aligned} w_{Rk}[n+1] &= w_{Rk}[n] - \frac{\mu}{2} \frac{\partial \sigma_\varepsilon^2}{\partial w_{Rk}} \\ w_{Ik}[n+1] &= w_{Ik}[n] - \frac{\mu}{2} \frac{\partial \sigma_\varepsilon^2}{\partial w_{Ik}} \qquad k = 0, 1, \ldots, P-1 \end{aligned}$$

.

Another expression for the gradient can be obtained by writing

$$\sigma_\varepsilon^2(\mathbf{w}) = \mathcal{E}\left\{|\varepsilon[n]|^2\right\} = \mathcal{E}\left\{\varepsilon[n](d[n] - \mathbf{w}^T\tilde{\boldsymbol{x}}[n])^*\right\}$$

The complex gradient of this expression is

$$\nabla_{\mathbf{w}^*}\sigma_\varepsilon^2 = -\mathcal{E}\left\{\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]\right\}$$

[Note that setting this form of the gradient to zero yields the *orthogonality principle*.] Appplying this result to (11.10) produces an alternative equation

$$\boxed{\mathbf{w}[n+1] = \mathbf{w}[n] + \mu\mathcal{E}\left\{\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]\right\}} \tag{11.12}$$

It will be seen that this is the more useful form for developing an adaptive filtering algorithm. The reason is obvious. In an adaptive filtering application the statisitical parameters $\mathbf{R}_{\boldsymbol{x}}$ and $\tilde{\mathbf{r}}_{d\boldsymbol{x}}$ in (11.11) are not known. (Otherwise the filter would not need to be adaptive.) The input $\tilde{\boldsymbol{x}}[n]$ and the error $\varepsilon[n]$ in (11.12) however, are assumed to be available.

11.2.2 Performance Analysis

For study of performance of the steepest descent algorithm it is best to work with the form (11.11). While $\mathbf{R}_{\boldsymbol{x}}$ and $\tilde{\mathbf{r}}_{d\boldsymbol{x}}$ may not be known, they clearly exist, and the performance is best understood in terms of these statistics. In particular, it will be seen that the covergence properties of the algorithm are dependent on the step size parameter μ and the eigenvalues of the correlation matrix.

Convergence of the Filter Weights

The time dependence of the weights, starting from an initial guess $\mathbf{w}[0]$, is the solution to the difference equation (11.11). Although it is not too difficult to write the explicit solution to this difference equation and examine its stability and convergence, it is much easier to define a residual weight vector

$$\mathbf{u}[n] = \mathbf{w}[n] - \mathbf{w}_\text{o} \tag{11.13}$$

which satisfies a simpler difference equation. Convergence of the residual weight vector to $\mathbf{0}$ is then equivalent to covergence of $\mathbf{w}$ to the optimum Wiener solution $\mathbf{w}_\text{o}$.

Solving (11.13) for $\mathbf{w}[n]$ and substituting this into (11.11) yields

$$\mathbf{u}[n+1] + \mathbf{w}_{\mathrm{o}} = \mathbf{u}[n] + \mathbf{w}_{\mathrm{o}} + \mu[\tilde{\mathbf{r}}_{d\boldsymbol{x}} - \mathbf{R}_{\boldsymbol{x}}(\mathbf{u}[n] + \mathbf{w}_{\mathrm{o}})]$$

Then canceling the common terms and using (11.6) to simplify, produces the result

$$\mathbf{u}[n+1] = (\mathbf{I} - \mu\mathbf{R}_{\boldsymbol{x}})\mathbf{u}[n] \tag{11.14}$$

which has the general solution

$$\mathbf{u}[n] = (\mathbf{I} - \mu\mathbf{R}_{\boldsymbol{x}})^n\mathbf{u}[0] \tag{11.15}$$

In order for the solution to converge to $\mathbf{0}$, therefore the matrix $(\mathbf{I} - \mu\mathbf{R}_{\boldsymbol{x}})^n$ must approach zero as n approaches infinity.

The conditions for this convergence can be obtained by first writing the correlation matrix in terms of its eigenvalues and eigenvectors:

$$\begin{aligned}\mathbf{R}_{\boldsymbol{x}} &= \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{*T} \\ &= \begin{bmatrix} | & | & & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_P \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_P \end{bmatrix} \begin{bmatrix} -- & \mathbf{e}_1^{*T} & -- \\ -- & \mathbf{e}_2^{*T} & -- \\ & \vdots & \\ -- & \mathbf{e}_P^{*T} & -- \end{bmatrix}\end{aligned} \tag{11.16}$$

where the columns of $\mathbf{E}$ are the eigenvectors $\mathbf{e}_i$ and λ_i are the eigenvalues. Then the matrix $(\mathbf{I} - \mu\mathbf{R}_{\boldsymbol{x}})$ can be written as

$$(\mathbf{I} - \mu\mathbf{R}_{\boldsymbol{x}}) = \mathbf{E}(\mathbf{I} - \mu\boldsymbol{\Lambda})\mathbf{E}^{*T}$$

where this follows from the equation above, and the fact that $\mathbf{E}$ is a *unitary* matrix, i.e., $\mathbf{E}\mathbf{E}^{*T} = \mathbf{E}^{*T}\mathbf{E} = \mathbf{I}$. Now by a repeated application of this formula while invoking the unitary property of $\mathbf{E}$, it is easy to show that

$$(\mathbf{I} - \mu\mathbf{R}_{\boldsymbol{x}})^n = \mathbf{E}(\mathbf{I} - \mu\boldsymbol{\Lambda})^n\mathbf{E}^{*T}$$

and thus (from (11.15))that

$$\mathbf{u}[n] = \mathbf{E}(\mathbf{I} - \mu\boldsymbol{\Lambda})^n\mathbf{E}^{*T}\mathbf{u}[0] \tag{11.17}$$

where $(\mathbf{I} - \mu\boldsymbol{\Lambda})^n$ is of the form

$$\begin{bmatrix} (1-\mu\lambda_1)^n & & & 0 \\ & (1-\mu\lambda_2)^n & & \\ & & \ddots & \\ 0 & & & (1-\mu\lambda_P)^n \end{bmatrix}$$

From (11.17) it is clear that $\mathbf{u}[n]$ will converge to $\mathbf{0}$ and $\mathbf{w}[n]$ will correspondingly converge to $\mathbf{w}_\mathrm{o}$ if and only if the elements of this diagonal matrix converge to zero as n approaches infinity. This will happen if and only if

$$|1 - \mu\lambda_i| < 1$$

for all i.

The last condition can be stated more simply as follows. First, since all of the λ_i are nonnegative, μ must be > 0. Secondly, since $\mu\lambda_i$ is now a positive quantity (or possibly 0), we must have $\mu\lambda_i < 2$ to satisfy the condition. Thus an equivalent condition for convergence, which is *necessary and sufficient*, is

$$\boxed{0 < \mu < \frac{2}{\lambda_{\max}} \qquad \text{CONDITION FOR CONVERGENCE} \qquad (11.18)}$$

where $\lambda_{\max}$ is the *largest* eigenvalue of $\mathbf{R}_{\boldsymbol{x}}$.

Modes of Convergence

A deeper understanding of the convergence of the weights to the Wiener solution can be obtained if the following change of variables is made:

$$\mathbf{v}[n] = \mathbf{E}^{*T}\mathbf{u}[n] = \mathbf{E}^{*T}(\mathbf{w}[n] - \mathbf{w}_\mathrm{o}) \qquad (11.19)$$

The components of $\mathbf{v}[n]$ represent projections of $\mathbf{u}[n]$ along the eigenvectors, i.e.,

$$\begin{bmatrix} v_1[n] \\ v_2[n] \\ \vdots \\ v_P[n] \end{bmatrix} = \begin{bmatrix} -- & \mathbf{e}_1^{*T} & -- \\ -- & \mathbf{e}_2^{*T} & -- \\ & \vdots & \\ -- & \mathbf{e}_P^{*T} & -- \end{bmatrix} \mathbf{u}[n]$$

and so represent a rotation of the coordinate system to one aligned with the eigenvectors of the correlation matrix. The relation between the coordinate systems defined by the weight vector $\mathbf{w}$ and the residual weight vector $\mathbf{u}$, and the rotated coordinate system defined by $\mathbf{v}$ is illustrated in Fig. 11.4. The ellipse represents a contour of the error surface $\sigma_\varepsilon^2(\mathbf{w})$ and, as will be shown later, the rotated coordinate system lies along the axes of this ellipse. Now from (11.17) we have

$$\mathbf{E}^{*T}\mathbf{u}[n] = (\mathbf{I} - \mu\mathbf{\Lambda})^n\mathbf{E}^{*T}\mathbf{u}[0]$$

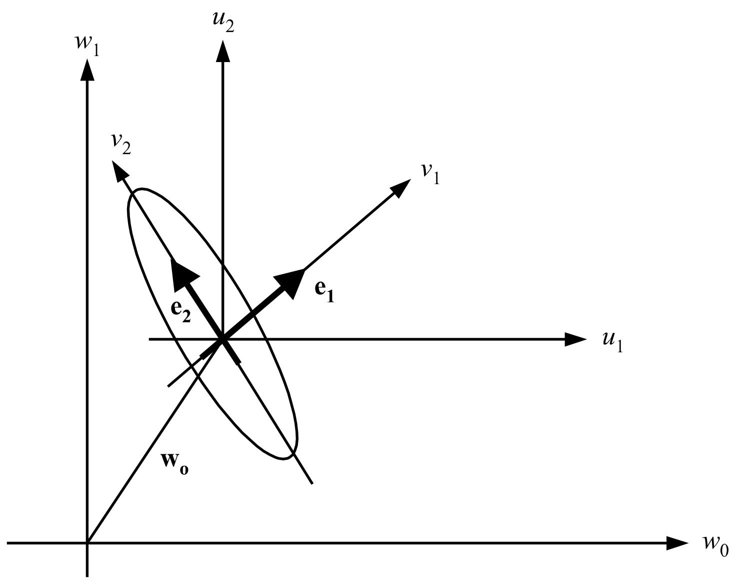


Figure 11.4: Coordinate systems defined by $\mathbf{w}$, $\mathbf{u}$, and $\mathbf{v}$.

or

$$\mathbf{v}[n] = (\mathbf{I} - \mu\mathbf{\Lambda})^n \mathbf{v}[0] \tag{11.20}$$

where, from (11.19)

$$\mathbf{v}[0] = \mathbf{E}^{*T}(\mathbf{w}[0] - \mathbf{w}_\mathrm{o}) \tag{11.21}$$

Because the matrix $(\mathbf{I} - \mu\mathbf{\Lambda})$ is diagonal, the components of $\mathbf{v}[n]$ are uncoupled and are referred to as *modes*. The time dependence of the modes is given by the scalar version of (11.20):

$$v_i[n] = (1 - \mu\lambda_i)^n v_i[0] \qquad i = 1, 2, \ldots, P \tag{11.22}$$

Now, by inverting (11.19), $\mathbf{u}[n]$ can be expressed in terms of the modes as

$$\mathbf{u}[n] = \mathbf{E}\mathbf{v}[n] = \begin{bmatrix} | \\ \mathbf{e}_1 \\ | \end{bmatrix} v_1[n] + \begin{bmatrix} | \\ \mathbf{e}_2 \\ | \end{bmatrix} v_2[n] + \cdots + \begin{bmatrix} | \\ \mathbf{e}_P \\ | \end{bmatrix} v_P[n]$$

Therefore, since $\mathbf{w}[n] = \mathbf{w}_\mathrm{o} + \mathbf{u}[n]$,

$$\mathbf{w}[n] = \mathbf{w}_\mathrm{o} + \sum_{i=1}^{P} \mathbf{e}_i v_i[n] = \mathbf{w}_\mathrm{o} + \sum_{i=1}^{P} \mathbf{e}_i v_i[0](1 - \mu\lambda_i)^n \tag{11.23}$$

The time dependence of the individual weights, which are the components of $\mathbf{w}$, is thus given by

$$w_k[n] = w_{\mathrm{o}k} + \sum_{i=1}^{P} e_{ki} v_i[0](1-\mu\lambda_i)^n \tag{11.24}$$

where $w_{\mathrm{o}k}$ is the k^{th} weight of the Wiener filter and e_{ki} is the $(k,i)^{\mathrm{th}}$ element of the eigenvector matrix $\mathbf{E}$.

Convergence Time for the Weight Vector

The covergence time of the i^{th} mode is measured by its time constant: the time that it takes the magnitude of the mode to decrease to $1/e$ (approximately 63%) of its initial value. The time constant τ_i of the i^{th} mode is thus defined by

$$|1-\mu\lambda_i|^{\tau_i} = e^{-1}$$

or

$$\tau_i = \frac{1}{-\ln|1-\mu\lambda_i|} \tag{11.25}$$

The time constant, as defined here, is a real number, which may be rounded to an integer.

To examine the convergence of the i^{th} mode more carefully, let us define

$$\alpha_i = 1-\mu\lambda_i$$

Convergence of the i^{th} mode occurs when $|\alpha_i| < 1$ which requires $0 < \mu < 2/\lambda_i$. The three regions of μ for which the mode converges are depicted in Fig. 11.5. For μ between 0 and $1/\lambda_i$, α_k is positive and the mode converges without oscillation. This is known as the *overdamped* condition. For μ between $1/\lambda_i$ and $2/\lambda_i$, α_i is negative and the mode also converges but this time with a change in sign at each time step, i.e., an oscillatory behavior. This is known as the *underdamped* condition. Finally, for $\mu = 1/\lambda_i$, α_i is equal to zero. This condition, if it could be achieved in real life, would cause the mode to converge in a single iteration regardless of its initial value! This is known as the *critically damped* condition.

It is shown in later sections of this chapter, that for the LMS algorithm – a practical adaptive filtering algorithm derived from the method of steepest descent – it is generally necessary to be conservative in the choice of μ.

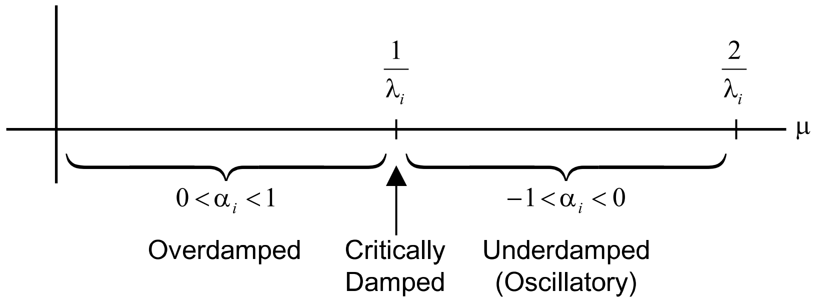


Figure 11.5: Regions of convergence for the k^{th} mode.

For that reason, and because it is often not desirable to have an oscillatory trajectory even if it is convergent, it is usually assumed that μ is chosen to be in the overdamped region for all modes (i.e., $\mu < 1/\lambda_{\max}$). In that case, one can drop the absolute value signs in (11.25) and write

$$\tau_i = \frac{1}{-\ln(1-\mu\lambda_i)} \lesssim \frac{1}{\mu\lambda_i} \qquad (11.26)$$

where the special symbol '$\lesssim$' is used here to mean "less than but approximately equal to" and holds for $\mu\lambda_i \ll 1$.

Since the time-dependent weight vector is a weighted sum of the modes (see (11.23)), convergence of the weight vector is dependent on convergence of all modes. Let us define the time constant $\tau_{\mathbf{w}}$ for the weight vector as the *largest* of the τ_i in (11.26). Then convergence time is thus determined by the *smallest* eigenvalue of the correlation matrix, that is,

$$\tau_{\mathbf{w}} = \frac{1}{-\ln(1-\mu\lambda_{\min})} \leq \frac{1}{\mu\lambda_{\min}}$$

Observe, however,that μ must satisfy (11.18) for convergence. If μ is written as

$$\mu = \rho\frac{2}{\lambda_{\max}}$$

where ρ is a fixed number between 0 and 1, then combination of the last two equations yields

$$\boxed{\tau_{\mathbf{w}} \leq \frac{1}{2\rho}\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)} \qquad 0 < \rho < 1$$

This last equation is another very important relation. It shows that time for convergence is dependent on the *eigenvalue spread* or the *condition number*

$$\chi \stackrel{\text{def}}{=} \frac{\lambda\text{max}}{\lambda_{\text{min}}} \tag{11.27}$$

of the input vector correlation matrix.

Convergence of the Mean-Square Error

Convergence of the weight vector to the Wiener filter solution signifies convergence of the mean-square error to its minimum value $\sigma^2_{\varepsilon_\text{o}}$. To examine this convergence in detail it is best to write (11.9) in terms of the vector $\mathbf{u}$ and ultimately in terms of $\mathbf{v}$ (see Fig. 11.4). Although this could be done by making direct substitutions in (11.9), a much easier method is as follows.The error corresponding to an arbitrary weight vector $\mathbf{w}$ can be written as

$$\begin{aligned}\varepsilon[n] &= d[n] - \mathbf{w}^T[n]\tilde{\boldsymbol{x}}[n] = d[n] - (\mathbf{w}_\text{o}^T + \mathbf{u}^T[n])\tilde{\boldsymbol{x}}[n] \\ &= \varepsilon_\text{o}[n] - \mathbf{u}^T[n]\tilde{\boldsymbol{x}}[n]\end{aligned}$$

where $\varepsilon_\text{o}[n]$ is the error for the optimal Wiener filter, defined by (11.8). Then

$$\begin{aligned}\sigma^2_\varepsilon &= \mathcal{E}\left\{|\varepsilon[n]|^2\right\} = \mathcal{E}\left\{(\varepsilon_\text{o}[n] - \mathbf{u}^T\tilde{\boldsymbol{x}}[n])(\varepsilon_\text{o}[n] - \mathbf{u}^T\tilde{\boldsymbol{x}}[n])^*\right\} \\ &= \mathcal{E}\left\{|\varepsilon_\text{o}[n]|^2\right\} + \mathbf{u}^T\mathcal{E}\left\{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^{*T}\right\}\mathbf{u}^*\end{aligned}$$

(The cross-terms disappear because of the *orthogonality principle.*) The mean-square error can therefore be written as[3]

$$\sigma^2_\varepsilon = \sigma^2_{\varepsilon_\text{o}} + \mathbf{u}^{*T}\mathbf{R}_{\boldsymbol{x}}\mathbf{u} \tag{11.28}$$

To simplify this further, (11.16 and (11.19) can be used to write

$$\mathbf{u}^{*T}\mathbf{R}_{\boldsymbol{x}}\mathbf{u} = \mathbf{u}^{*T}\mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{*T}\mathbf{u} = \mathbf{v}^{*T}\boldsymbol{\Lambda}\mathbf{v}$$

so that (11.28) can be written as

$$\begin{aligned}\sigma^2_\varepsilon &= \sigma^2_{\varepsilon_\text{o}} + \mathbf{v}^{*T}\boldsymbol{\Lambda}\mathbf{v} \\ &= \sigma^2_{\varepsilon_\text{o}} + \sum_{i=1}^{P}\lambda_i|v_i|^2\end{aligned} \tag{11.29}$$

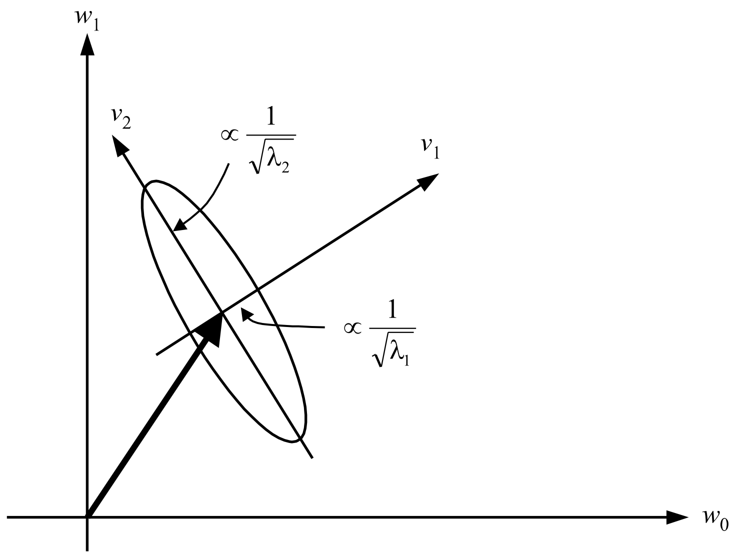

w_1
v_2
$\propto \frac{1}{\sqrt{\lambda_2}}$
v_1
$\propto \frac{1}{\sqrt{\lambda_1}}$
w_0

Figure 11.6: Contours of error surface $\sigma_\varepsilon^2 =$ constant.

The contours of constant σ_ε^2 are *ellipsoids* with axes proportional to $1/\sqrt{\lambda_i}$ (see Fig. 11.6). So, note that unlike the contours of the Gaussian density function, the *smallest* eigenvalue determines the largest dimension of the ellipse.

The time dependence of the mean-square error can be made explicit by substituting the expression for the modes (11.22) in (11.29), thus obtaining

$$\sigma_\varepsilon^2[n] = \sigma_{\varepsilon_o}^2 + \sum_{i=1}^{P} \lambda_i (1-\mu\lambda_i)^{2n} |v_i[0]|^2$$

where the $v_i[0]$ are computed from (11.21). From this it can be seen that the time constant for the mean-square error is upper bounded by

$$\tau_{\text{mse}} \leq \frac{1}{-2\ln(1-\mu\lambda_{\text{min}})} \lesssim \frac{1}{2\mu\lambda_{\text{min}}} \tag{11.30}$$

A plot of $\sigma_\varepsilon^2[n]$ versus n is known as the *learning curve*. A typical learning curve is depicted in Fig. 11.7.

[3]See footnote on page 8.

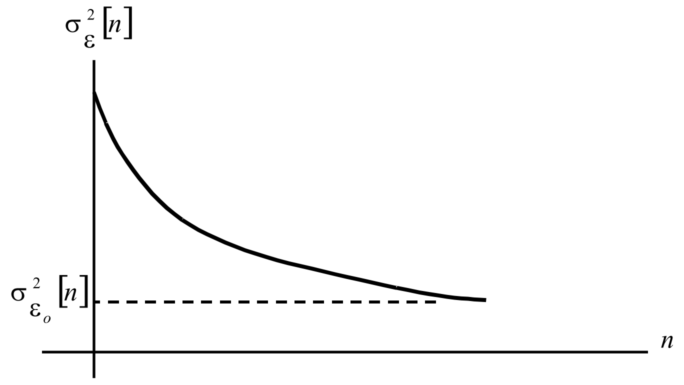


Figure 11.7: Learning curve for the Steepest Descent algorithm.

11.3 THE LMS ALGORITHM

11.3.1 A Simple Adaptive Algorithm

The steepest descent method as given by (11.11) or (11.12) is not in itself a suitable algorithm for adaptive filtering because it involves quantities which are not assumed to be known. A simple adaptive algorithm can be derived from (11.12) however by ignoring the expectation operator and replacing the expectation by the quantity itself. The resulting algorithm is known as the Least Mean Squares (LMS) algorithm [7] and has the form

$$\boxed{\begin{array}{rcll} \varepsilon[n] & = & d[n] - \mathbf{w}^T[n]\tilde{\boldsymbol{x}}[n] & \text{(a)} \\ \mathbf{w}[n+1] & = & \mathbf{w}[n] + \mu\,\varepsilon[n]\tilde{\boldsymbol{x}}^*[n] & \text{(b)} \end{array} \qquad \text{LMS} \qquad (11.31)}$$

Note from the discussion preceding equation 11.11 that this is equivalent to changing the tap weights in the direction of the negative gradient of the *instantaneous* squared error, i.e.,

$$\nabla_{\mathbf{w}^*}|\varepsilon(n)|^2 = -\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]$$

This quantity is a random variable and is not guaranteed to lower the error at each iteration. However when the condition (11.18) for convergence of the steepest decent algorithm is satisfied, the LMS algorithm also converges in a manner to be discussed shortly. Moreover, it is extremely simple to apply and requires only $\mathcal{O}(P)$ arithmetic operations per iteration. An unresolved issue however, is how to choose the step size parameter μ. Evidently, at least (11.18) must be satisfied; but we do not presume to know $\mathbf{R}_{\boldsymbol{x}}$ and its eigenvalues. A practical solution is to use a more conservative bound in place of (11.18) that can be easily estimated. Since the trace of $\mathbf{R}_{\boldsymbol{x}}$ is equal to the sum of its eigenvalues, and these eigenvalues are all nonnegative, it is clear that

$$\lambda_{\max} \leq \text{tr } \mathbf{R}_{\boldsymbol{x}}$$

Substituting this into (11.18) leads to the sufficient condition:

$$\boxed{0 < \mu < \frac{2}{\text{tr } \mathbf{R}_{\boldsymbol{x}}} = \frac{2}{\text{mean tap input power}} = \frac{2}{PR_x[0]} \qquad (11.32)}$$

This provides a starting point for selecting μ since the average power ($R_x[0]$) can be easily estimated from the input data. In practice, the chosen value for μ is often in the range of 0.1 to 0.5 times the upper bound, or even less. The trade-offs involving this are discussed in the next section.

A simple example in terms of a system identification problem is illustrated below. Several important points discussed in this chapter are pointed out as well as some new phenomena which will need to be further explored.

Example 11.1 An illustration of the LMS method for system identification (see Fig. 11.1 (a)) is considered here. In this experiment the unknown system is described by

$$y[n] = x[n] + 0.7x[n-1]$$

A first order adaptive filter is used ($P = 2$), so that the weights w_0 and w_1 should converge to 1.0 and 0.7 respectively. The input sequence $x[n]$ is chosen to be a first order process of the form

$$x[n] = ax[n-1] + w[n]$$

where $w[n]$ is white noise with variance $\sigma_w^2 = 1$. The parameter a determines the eigenvalue spread of the input process (see Prob. 11.1). Some values of a, the corresponding eigenvalue spread χ, and the upper bound ($2/\lambda_{\max}$) on μ are given in the table below.

a	χ	*upper bound*
0	1	2
0.25	1.667 (1.59)	1.5 (1.54)
0.5	3 (2.8)	1 (1.04)
0.75	7 (6.3)	0.5 (0.54)
0.95	39 (31)	0.1 (0.12)

The main numbers listed are the theoretical values. The experimental values estimated from the data are shown in parentheses for comparison. The first case listed $a = 0$ corresponds to pure white noise and is the ideal case for rapid convergence.

Figure 11.8 shows the trajectories of the filter coefficients for different values of a. In each case, μ was taken to be $\frac{1}{10}$ of the upper bound for that case. Notice that the settling time increases as a and thus the eigenvalue spread increases. In Fig. 11.8 (a) and (b) the filter coefficients have converged to the correct values (shown by the dotted lines) within less than 15 iterations. In Fig. 11.8 (c) ($\chi = 7$) about 100 iterations are required for convergence, while in (d) ($\chi = 39$) the coefficients still have not reached their final values in 500 iterations.

$\square$

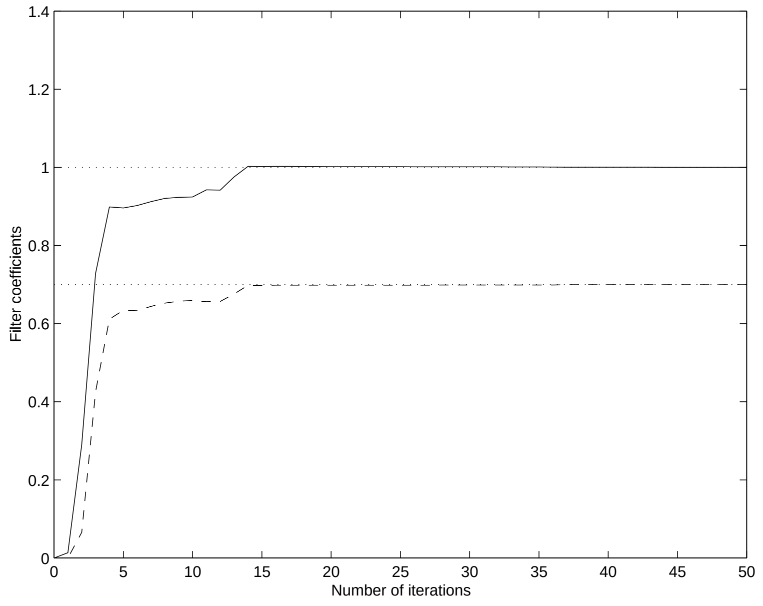
From this example and the previous discussions, it is clear that the LMS algorithm performs best when the input is a white noise sequence. Thus techniques that attempt to orthogonalize the input while equalizing the input power may be used in some applications. Some of these techniques are discussed later.

The experimental learning curve (i.e., squared error) for the case depicted in Fig. 11.8(d) of the example is shown in Fig. 11.9. Note that although the squared error is decreasing very slowly it is not nearly as low as the value for the Wiener filter, which is approximately -285 dB.[4] It turns out, that the error will eventually fluctuate around some average steady state value, but this value will *never* be quite equal to the Wiener filter error. This is known as the "misadjustment" effect of the LMS algorithm. The reason for it is explored in the next section.

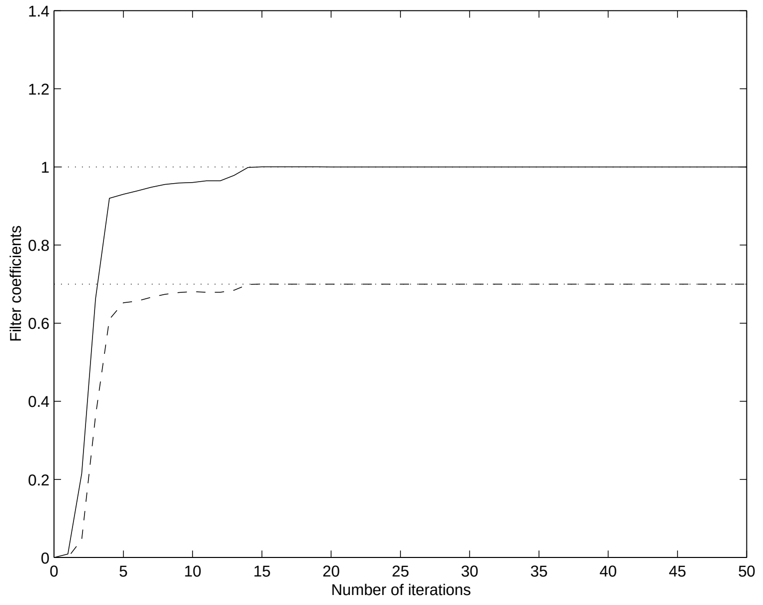
11.3.2 Discussion of LMS Convergence

A block diagram for the LMS adaptive algorithm (11.31) is given in Fig. 11.10. The algorithm looks deceptively simple, but there is an underlying complexity that makes the algorithm extremely difficult to analyze. This complexity arises from a number of characteristics. First, although the method is based on an FIR filter, the coefficients are not constant, so the overall system in Fig. 11.10 is *time-varying.* Secondly, since the error is fed

[4]This value, although ridiculously low for any physical experiment, is really not unreasonable for this simulation where there has been no observation noise added to the output of the linear system.

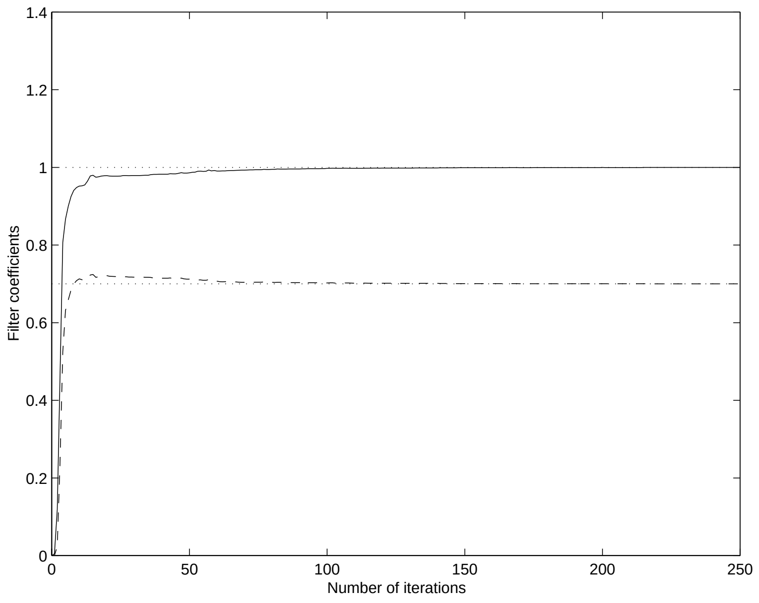


(a)

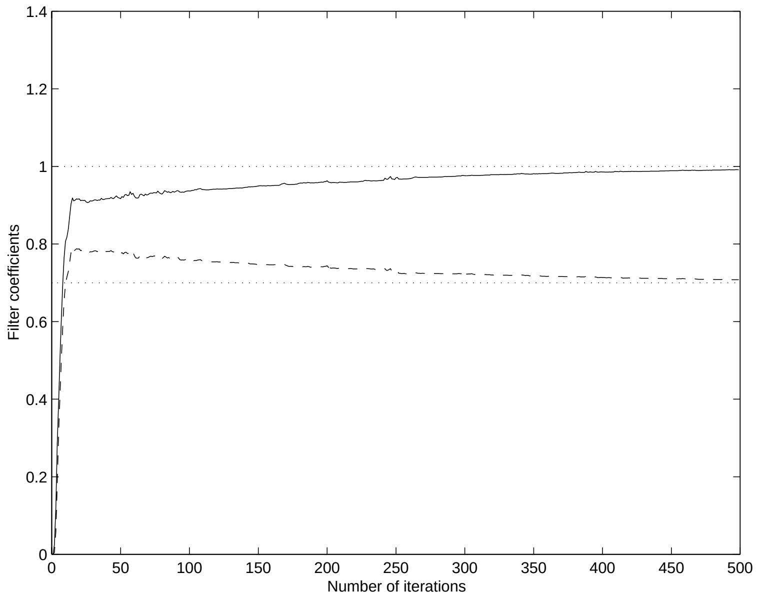


(b)

Figure 11.8: LMS system identification example. (a) $a = 0.25, \chi = 1.667, \mu = 0.15$ (b) $a = 0.5, \chi = 3, \mu = 0.1$.



(c)



(d)

Figure 11.8: LMS system identification example, cont'd. (c) $a = 0.75, \chi = 7, \mu = 0.05$ (d) $a = 0.95, \chi = 39, \mu = 0.01$

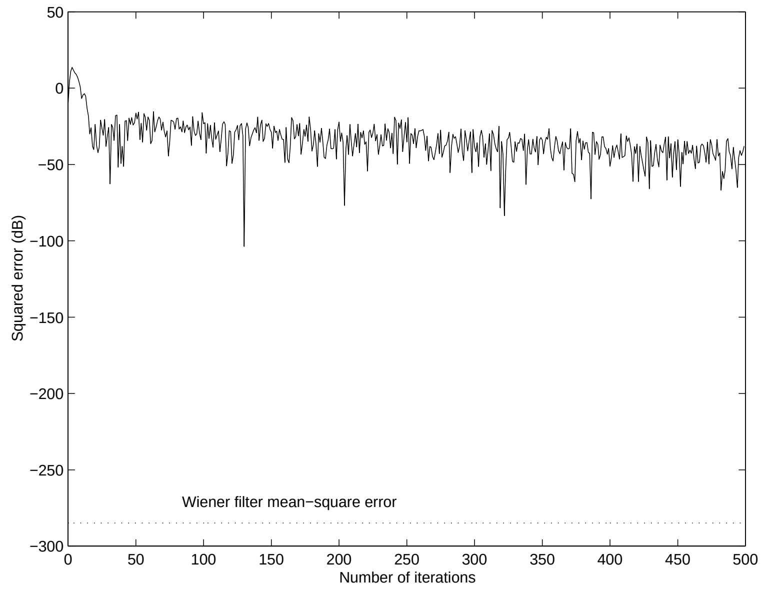


Figure 11.9: Learning curve for the LMS system identification example. $(a = 0.95, \chi = 39, \mu = 0.01)$

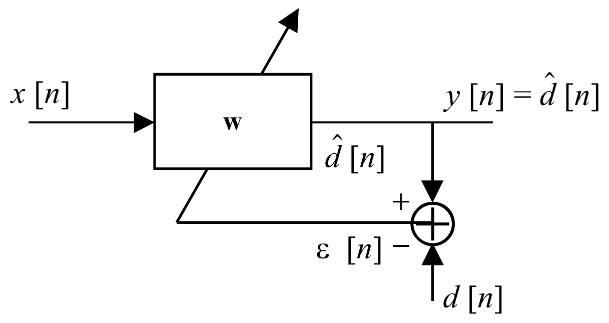


Figure 11.10: Block diagram for LMS adaptive filter.

back and used to update the filter coefficients, the overall system is *nonlinear.* This is evident if you substitute (11.5) into (11.31) to obtain

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu\,\tilde{\boldsymbol{x}}^*[n](d[n] - \tilde{\boldsymbol{x}}^T[n]\mathbf{w}[n]) \tag{11.33}$$

Since the filter coefficients are nonlinear functions of the input, the system output is also a nonlinear function of the input. Last, but not least, the input $x[n]$ is a random process. Therefore the system in Fig. 11.10 is described by a *stochastic nonlinear* difference equation. The input, the gradient estimate, and the filter coefficients are all *random processes.*

Aside from the complexity inherent in the system, there is the issue of convergence for a random process. Two types of stochastic convergence will be considered here. The first type is called *convergence of the mean.*[5] This type of convergence states that

$$\lim_{n\to\infty} \mathcal{E}\{\varepsilon[n]\} = \varepsilon_\infty \quad \text{and} \quad \lim_{n\to\infty} \mathcal{E}\{\mathbf{w}[n]\} = \mathbf{w}_\infty$$

where ε_∞ and $\mathbf{w}_\infty$ are finite constant limiting values. Although this type of convergence can be shown for LMS, it is of marginal value by itself. In particular, although the mean value of the error and the filter coefficients may become constant, the *actual* values of these variables can continue to jump around. In fact, this jumpy behavior is exactly what the LMS algorithm tends to produce.

The other type of convergence is called *convergence in mean-square*, which states that

$$\lim_{n\to\infty} \mathcal{E}\left\{|\varepsilon[n]|^2\right\} = \sigma_\varepsilon^2$$

where σ_ε^2 is a finite constant. This is a stronger kind of convergence which establishes the variance of the error as $n \to \infty$. This type of convergence can also be established for the LMS algorithm although a correct proof is difficult mathematically and has only recently been known (see [8]). The result implies however, that the jumpy behavior can be kept managable by a proper selection of the step size parameter.

Since a formal proof of any type of convergence is difficult, most treatments use a set of generally unrealistic assumptions known as "independence theory" (see Problems 11.2 and 11.3) to avoid these difficulties. Rather than follow along those lines here, let us proceed to examine the problem in more

[5] Also called convergence in mean value.

detail and try to infer something about its limiting characteristics. Although no actual "proofs" of convergence are provided, the arguments should be sufficient to explain the general performance of the algorithm.

To begin, consider the vector $\mathbf{u}$ defined by (11.13). Since $\mathbf{w}$ is now a random vector, $\mathbf{u}$ is also a random vector and represents the *deviation* of the filter coefficients from those of the Wiener optimal filter. By substituting (11.13) into (11.33) and rearranging, we can obtain a difference equation for the deviation $\mathbf{u}$ as

$$\mathbf{u}[n+1] = \left(\mathbf{I} - \mu\tilde{\boldsymbol{x}}^*[n]\tilde{\boldsymbol{x}}^T[n]\right)\mathbf{u}[n] + \mu\tilde{\boldsymbol{x}}^T[n]\varepsilon_{\text{o}}[n] \tag{11.34}$$

where $\varepsilon_{\text{o}}[n]$ is the Wiener filter error sequence given by (11.8). The corresponding equation for the filter error sequence is obtained by substituting (11.13) into (11.5) and using (11.8). The result is

$$\varepsilon[n] = \varepsilon_{\text{o}}[n] - \tilde{\boldsymbol{x}}^T[n]\mathbf{u}[n] \tag{11.35}$$

These last two equations can be used to infer some general characteristics of the LMS algorithm. First, examine (11.34) and observe that this difference equation is "driven" by the term on the right hand side involving $\varepsilon_{\text{o}}[n]$. Therefore unless $\varepsilon_{\text{o}}[n]$ is zero (which it is not), or $\tilde{\boldsymbol{x}}[n] = \mathbf{0}$ (unlikely), $\mathbf{u}[n]$ will not approach $\mathbf{0}$ as a steady-state solution. In fact, since $\varepsilon_{\text{o}}[n]$ is a white noise process, $\mathbf{u}[n]$ will not tend to approach *any* constant value, but will continue to vary randomly with time. Thus $\mathbf{w}$ (which equals $\mathbf{u}+\mathbf{w}_{\text{o}}$) will not tend to a steady-state value of $\mathbf{w}_{\text{o}}$ but will continue to vary in time. Note also that μ acts like a gain factor in (11.34). Smaller values of μ lead to smaller variations of $\mathbf{u}[n]$. This is an important fact to remember.

In spite of the somewhat erratic behavior it can be shown that $\mathcal{E}\{\mathbf{u}[n]\}$ converges to $\mathbf{0}$ and thus $\mathbf{w}[n]$ satisfies *convergence of the mean* to $\mathbf{w}_{\text{o}}$. This is plausible when you consider that $\mathcal{E}\{\tilde{\boldsymbol{x}}^T[n]\varepsilon_{\text{o}}[n]\} = \mathbf{0}$ by the Orthogonality Principle; thus the difference equation for $\mathcal{E}\{\mathbf{u}[n]\}$ obtained by taking the expected value of (11.34) has no driving term.

Now turn to (11.35) and notice that since $\varepsilon[n]$ consists of the sum of two terms, the variance of $\varepsilon[n]$ is more than just the variance of $\varepsilon_{\text{o}}[n]$. Thus the mean-square error of the LMS algorithm will always be greater than that of the Wiener filter. In other words, the difference, which is known as the *excess error*, will always be greater than zero. It is apparent that the excess error is also controlled by the step size parameter however, so that smaller values of μ tend to reduce the excess error.

In the literature on adaptive filters the excess error is usually represented as a normalized quantity called the *misadjustment*, and defined by

$$\mathcal{M} \overset{\text{def}}{=} \frac{\sigma_\varepsilon^2 - \sigma_{\varepsilon_o}^2}{\sigma_{\varepsilon_o}^2} = \frac{\sigma_\varepsilon^2}{\sigma_{\varepsilon_o}^2} - 1 \tag{11.36}$$

An approximate expression for the misadjustment derived using the independence theory can be given as

$$\mathcal{M} = \sum_{i=1}^{P} \frac{\mu\lambda_i}{2 - \mu\lambda_i}$$

(see Problem 11.3). The existence of excess error or misadjustment is an inherent characteristic of the LMS algorithm. Increasing the value of μ tends to increase the rate of convergence but also to increase the misadjustment, and *vice versa*.

11.3.3 Related Forms

A number of algorithms related to the basic LMS algorithms are possible. Some are intended to improve performance of the basic algorithm (at the expense of modest increases in computation) while others, such as the sign algorithms, aim at reducing the computation to a bare minimum. This section cites some of the more well-known variations of LMS.

Leaky LMS

Let us return for a moment to the method of steepest descent, on which the LMS algorithm is based. Recall that the tap weight vector is given by the iteration formulas (11.11) and (11.12), which are summarized here for convenience:

$$\begin{aligned} \mathbf{w}[n+1] &= \mathbf{w}[n] + \mu\mathcal{E}\{\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]\} \\ &= \mathbf{w}[n] + \mu(\tilde{\mathbf{r}}_{d\boldsymbol{x}} - \mathbf{R}_{\boldsymbol{x}}\mathbf{w}[n]) \end{aligned}$$

Based on this last expression, it was found that the modes of the problem converged according to $(1-\mu\lambda_i)^n$, where the λ_i are the eigenvalues of the correlation matrix.

It is instructive to consider what happens if one or more of the eigenvalues is equal to zero. Then it is clear that the mode remains constant with time and does not converge. Worst yet, suppose that due to numerical inaccuracies some of these “zero” eigenvectors are computed as very small *negative* numbers. In this case, the modes actually diverge! To prevent these undesirable results, numerical analysists have suggested a technique called “leakage,” where the weight iteration is modified to the following expression

$$\begin{aligned}\mathbf{w}[n+1] &= (1-\mu\alpha)\mathbf{w}[n] + \mu(\tilde{\mathbf{r}}_{d\boldsymbol{x}} - \mathbf{R}_{\boldsymbol{x}}\mathbf{w}[n]) \\ &= \mathbf{w}[n] + \mu\left(\tilde{\mathbf{r}}_{d\boldsymbol{x}} - (\mathbf{R}_{\boldsymbol{x}} + \alpha\mathbf{I})\mathbf{w}[n]\right)\end{aligned}$$

with α (the leakage coefficient) being a small positive number much less than one. Evidently, the correlation matrix has been replaced by a modified correlation matrix $\mathbf{R}_{\boldsymbol{x}} + \alpha\mathbf{I}$ that has no zero eigenvalues. Therefore convergence is assured if μ is chosen to satisfy

$$0 < \mu < \frac{2}{\alpha + \lambda_{\max}}$$

The corresponding LMS algorithm takes the form

$$\boxed{\mathbf{w}[n+1] = (1-\mu\alpha)\mathbf{w}[n] + \mu\,\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]} \tag{11.37}$$

which is known as the *leaky LMS algorithm.*

Unfortunately, the underlying leaky steepest descent algorithm does *not* converge to the Wiener-Hopf solution, but instead to the modified solution

$$\mathbf{w} = (\mathbf{R}_{\boldsymbol{x}} + \alpha\mathbf{I})^{-1}\tilde{\mathbf{r}}_{d\boldsymbol{x}}$$

Thus the leaky LMS algorithm has an additional bias.

The leaky LMS algorithm can also be derived by minimization of the weighted sum

$$|\varepsilon[n]|^2 + \alpha\|\mathbf{w}[n]\|^2$$

(see Problem 11.6). Note also that the modified correlation matrix that appears in the leaky version of the algorithm is the correlation matrix for the input plus a small amount of white noise. Thus the same effect can be achieved by actually adding a small amount of white noise with variance α to the input of the LMS adaptive filter.

Normalized LMS

Another version of the LMS algorithm, known as *Normalized LMS*, can be derived based on the following considerations. At time $n+1$ the weight vector is updated by adding to it some amount $\boldsymbol{\delta}$. That is,

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \boldsymbol{\delta}[n+1]$$

The time-varying step $\boldsymbol{\delta}$ is chosen to satisfy the condition

$$\mathbf{w}^T[n+1]\tilde{\boldsymbol{x}}[n] = d[n] \tag{11.38}$$

and $\|\boldsymbol{\delta}\|^2$ is minimized subject to the above constraint.

Note that the constraint does *not* say $\mathbf{w}^T[n+1]\tilde{\boldsymbol{x}}[n+1] = d[n+1]$, as this would state that the filter is perfect! The constraint does require the new weight to estimate the *present* data perfectly however, and to make the smallest possible change to do so.

The minimization is straightforward using the techniques outlined in Appendix A and is given as a problem (see Problem 11.7). The optimal $\boldsymbol{\delta}$ is given by

$$\boldsymbol{\delta}[n+1] = \frac{\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]}{\|\tilde{\boldsymbol{x}}[n]\|^2}$$

leading to the algorithm

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \frac{\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]}{\|\tilde{\boldsymbol{x}}[n]\|^2} \tag{11.39}$$

Let us discuss this result briefly. Equation 11.32 shows that a conservative bound for the step size parameter μ in the LMS algorithms involves the reciprocal of the mean tap input power $PR_x[0]$. An intuitively reasonable estimate for this quantity would be the *instantaneous* tap input power, which is the squared norm of the input vector, i.e.,

$$\hat{P}_T[n] = \|\tilde{\boldsymbol{x}}[n]\|^2 = \sum_{k=n-P+1}^{n} |x[k]|^2 \tag{11.40}$$

This estimate is *unbiased* since $\mathcal{E}\left\{\hat{P}_T[n]\right\} = PR_x[0]$. With this estimate (11.39) is equivalent to the ordinary LMS algorithm with μ taken exactly half way between its upper and lower bounds (see (11.32)).

The algorithm known as "Normalized LMS" modifies (11.39) in two ways. First it introduces a a normalized step size parameter which is theoretically in the range $0 < \mu' < 2$ (but see below). Secondly, it adds a small positive number ϵ (not to be confused with the error sequence $\varepsilon[n]$) to the vector norm $\|\tilde{\boldsymbol{x}}[n]\|^2$. This is intended to prevent instability, should the actual norm of the input vector become very small. The resulting Normalized LMS algorithm is given by

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \frac{\mu'}{\epsilon + \|\tilde{\boldsymbol{x}}[n]\|^2}\,\varepsilon[n]\tilde{\boldsymbol{x}}^*[n] \tag{11.41}$$

To avoid the P additions per iteration that normally would be needed to compute the vector norm, a sliding window computation can be used. Equation (11.41) is then expressed as two equations:

$$\boxed{\mathbf{w}[n+1] = \mathbf{w}[n] + \frac{\mu'}{\hat{P}_T[n]}\,\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]} \tag{11.42}$$

and

$$\boxed{\hat{P}_T[n] = \hat{P}_T[n-1] + |x[n]|^2 - |x[n-P]|^2} \tag{11.43}$$

(This last equation follows easily from (11.40).) The recursion (11.43) is initialized by setting $\hat{P}_T[-1] = \epsilon$ and providing data or zeros for $x[-1] \ldots x[-P]$.

The normalized LMS algorithm has a number of potential advantages over the basic LMS algorithm (11.31):

1. There is no need for separate computation of an upper bound on the step size parameter; μ' is independent of the tap input power. In practice, this parameter is usually chosen in the range $0.1 \leq \mu' < 1$.

2. The normalization quantity $\hat{P}_T[n]$ is estimated directly from the input data.

3. Because $\hat{P}_T[n]$ is computed from the *most recent* data, the normalization follows changes in the input statistics.

The last property allows the normalized step size parameter μ' to remain fixed under changing statistics and can result in better "tracking" of nonstationary processes. On the other hand, the normalization is performed at each step and requires a few additional arithmetic operations.

LMS with Filtered Normalization

The estimate $\hat{P}_T[n]$ given in (11.40) and (11.43) is a moving sum estimate of the mean tap input power. It is also possible to use a weighted sum computed as

$$\boxed{\hat{P}_T[n] = \beta\hat{P}_T[n-1] + P(1-\beta)|x[n]|^2} \tag{11.44}$$

where β is a weighting factor in the range $0 < \beta < 1$. Equation 11.44 is sometimes refered to as a *filtered* estimate. Using (11.42) in conjuction with (11.44) provides an alternative form of normalization for the LMS algorithm.

Equation 11.44 uses all the data in the tap power computation, but gives decreasing weight to earlier data. This can be seen clearly by writing the solution to (11.44) in closed form as[6]

$$\hat{P}_T[n] = P(1-\beta)\sum_{k=0}^{n}\beta^k|x[n-k]|^2 \tag{11.45}$$

Thus $|x[n]|^2$ has unity weight and $|x[n-k]|^2$ has weight β^k, which decreases with k.

The weighted estimate $\hat{P}_T[n]$ is an *asymptotically unbiased* estimate of the mean tap input power. To see this, take the expectation of (11.44) to write

$$\mathcal{E}\left\{\hat{P}_T[n]\right\} = \beta\mathcal{E}\left\{\hat{P}_T[n-1]\right\} + P(1-\beta)\underbrace{\mathcal{E}\left\{|x[n]|^2\right\}}_{R_x[0]}$$

If $\hat{P}_T[n]$ approaches a limit P_T as $n \to \infty$ then the last equation becomes

$$P_T = \beta P_T + P(1-\beta)R_x[0]$$

which has a solution $P_T = PR_x[0]$. That the limiting value does exist and is given by $P_T = PR_x[0]$ can be shown more rigorously by applying the expectation to (11.45) (see Problem 11.8).

Sign Algorithms

In applications involving *real-valued* signals, where low computation makes the difference between feasibility and infeasibility of an application, the LMS

[6]Equation 11.44 is simply a first order difference equation with impulse response $P(1-\beta)\beta^n$ and (11.45) just represents the convolution of this sequence with the "input" sequence $|x[n]|^2$.

algorithm can be further simplified to use only the *sign* of some of the terms. A basic form of sign algorithm is known as the *sign-error* algorithm and has the weight update equation

$$\boxed{\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \,\mathrm{sgn}\left\{\varepsilon[n]\right\} \tilde{\boldsymbol{x}}[n]} \tag{11.46}$$

where the function sgn$\{\cdot\}$ is $+1$ if its argument is positive and -1 if the argument is negative. This algorithm can be shown to minimize the absolute value of the instantaneous error, i.e., $|\varepsilon[n]|$ (see Problem 11.9); therefore it is also called the *Least Mean Absolute Value* (LMAV) algorithm. Note that since the weight update formula uses only the sign of the error, this algorithm moves the weights in the same *direction* as the standard LMS algorithm, but the magnitude of the change is different.

This approach can be taken one step further by using just the sign of each component of the observation vector as well as the sign of the error. The resulting *sign-sign* algorithm thus takes the form

$$\boxed{\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \,\mathrm{sgn}\left\{\varepsilon[n]\right\} \mathrm{sgn}\left\{\tilde{\boldsymbol{x}}[n]\right\}} \tag{11.47}$$

The price paid for the simplicity of the sign algorithms is generally slower convergence and a greater tendency toward instability than the standard LMS algorithm. Stability can be improved, as in the standard LMS by adding a leakage term, that is, multiplying the term $\mathbf{w}[n]$ on the right side of (11.46) or (11.47) by $(1-\mu\alpha)$ where α is the leakage coefficient.

Quasi-Newton Methods

The LMS method is derived from the steepest descent technique in numerical analysis and optimization, which uses only first derivatives of the error surface. A well established way to accelerate convergence, is to use both the gradient and the Hessian (matrix of second derivatives) of the error surface in the iteration. The basic technique for this procedure is called the Gauss-Newton method. In the application of this method to the FIR filtering problem, the Hessian turns out to be the inverse of the input autocorrelation matrix, and the corresponding stochastic iteration takes the form

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \,\varepsilon[n] \mathbf{R}_{\tilde{\boldsymbol{x}}}^{-1} \tilde{\boldsymbol{x}}^*[n]$$

This equation seems to defeat the purpose of an adaptive iterative algorithm since it requires inversion of the correlation matrix, which is equivalent to solving the Wiener-Hopf equation. However, if the true inverse can be replaced by an approximation $\hat{\mathbf{R}}_{\boldsymbol{x}}^{-1}$, which is either fixed or low in computational complexity, the above algorithm becomes viable. Such algorithms have been called *quasi-Newton* algorithms [6].

The quasi-Newton algorithm can be interpreted in a number of ways. Intuitively the scalar step-size parameter μ in the LMS algorithm is replaced by a scaled matrix $\mu\hat{\mathbf{R}}_{\boldsymbol{x}}^{-1}$ which allows more degrees of freedom. In addition, the incorporation of that matrix can be interpreted as a whitening transformation applied to the input. Our study of convergence for the steepest descent and LMS algorithms showed that a white input process is the optimum situation.

A number of approximations are possible. Marshall and Jenkins [9] have developed an approach called Fast Quasi-Newton which has computational requirements of $\mathcal{O}(P^2)$. Also, the well-known Recursive Least Squares (RLS) algorithm (described later) is essentially of this form and has similar computational requirements, although it is not derived as a stochastic optimization problem. Finally fixed matrices with just a few parameters, such as a diagonal or tri-diagonal matrix or matrices derived from an approximate whitening transformation such as the discrete cosine transform can all be applied to develop a method. Computational requirements will typically vary from $\mathcal{O}(P)$ to $\mathcal{O}(P^2)$.

11.4 LMS FOR LATTICES

The ability to represent a prediction error filter in a lattice form suggests that it may be possible to form an adaptive algorithm for *linear prediction* using the lattice realization. Further, the ability to formulate FIR Wiener filters in a lattice form (See Chapter 8, Section 8.12) suggests that perhaps *all* FIR adaptive filters may be implemented in lattice form.

This section begins by considering the adaptive linear prediction problem, with a filter implemented in lattice form. Once the appropriate form of the algorithms is developed, we move on to consider the so-called “joint process estimation” problem and the adaptive lattice structure that can be applied to all FIR filters.

11.4.1 Adaptive Linear Prediction

The LMS method for lattice filters is known as the Gradient Adaptive Lattice (GAL) algorithm developed by Griffith [10, 11]. The algorithm is slightly more complex than the basic LMS algorithm, as it involves a *dual* recursion: a recursion in filter *order* as well as a recursion in time. You may notice that its development bears some similarity to that of the Burg algorithm described in Chapter 9.

We begin by considering a lattice prediction error filter of order P and assume that the reflection coefficients up to order $p-1$ have been determined. The reflection coefficient γ_p can be found by minimizing the mean sum of squared forward and backward prediction errors, denoted by

$$J_p = \mathcal{E}\left\{|\varepsilon_p[n]|^2 + |\varepsilon_p^b[n]|^2\right\} \tag{11.48}$$

Let us first show that minimization of this quantity with respect to γ_p determines the correct value for the reflection coefficient in a standard non-adaptive linear prediction problem. The forward and backward prediction errors satisfy the following recursion (derived in Chapter 8)

$$\boxed{\begin{array}{ll} \varepsilon_p[n] = \varepsilon_{p-1}[n] - \gamma_p^*\varepsilon_{p-1}^b[n-1] & (a) \\ \varepsilon_p^b[n] = \varepsilon_{p-1}^b[n-1] - \gamma_p\varepsilon_{p-1}[n] & (b) \end{array}} \tag{11.49}$$

Substituting (11.49) into (11.48) results in

$$\begin{aligned} J_p = \mathcal{E}\Big\{&\left[\varepsilon_{p-1}[n] - \gamma_p^*\varepsilon_{p-1}^b[n-1]\right]\left[\varepsilon_{p-1}^*[n] - \gamma_p\varepsilon_{p-1}^{b*}[n-1]\right] \\ &+ \left[\varepsilon_{p-1}^b[n-1] - \gamma_p\varepsilon_{p-1}[n]\right]\left[\varepsilon_{p-1}^{b*}[n-1] - \gamma_p^*\varepsilon_{p-1}^*[n]\right]\Big\} \end{aligned}$$

and taking the complex gradient produces

$$\begin{aligned} \nabla_{\gamma_p^*} J_p = -\mathcal{E}\Big\{&\varepsilon_{p-1}^b[n-1]\left[\varepsilon_{p-1}^*[n] - \gamma_p\varepsilon_{p-1}^{b*}[n-1]\right] \\ &+ \left[\varepsilon_{p-1}^b[n-1] - \gamma_p\varepsilon_{p-1}[n]\right]\varepsilon_{p-1}^*[n]\Big\} \end{aligned} \tag{11.50}$$

The optimal reflection coefficient is then found by setting (11.50) to zero, which yields

$$\gamma_p = \frac{\mathcal{E}\left\{\varepsilon_{p-1}^b[n-1]\varepsilon_{p-1}^*[n]\right\}}{\frac{1}{2}\left[\mathcal{E}\left\{|\varepsilon_{p-1}[n-1]|^2\right\} + \mathcal{E}\left\{|\varepsilon_{p-1}^b[n-1]|^2\right\}\right]}$$

Since the expected value of the forward and backward squared errors is the same, this last result can be written as

$$\gamma_p = \frac{\mathcal{E}\left\{\varepsilon^b_{p-1}[n-1]\varepsilon^*_{p-1}[n]\right\}}{\mathcal{E}\left\{|\varepsilon^b_{p-1}[n-1]|^2\right\}}$$

which is the same expression derived in Chapter 8, Section 8.7. We conclude, therefore, that the proposed procedure of minimizing (11.48) with respect to the reflection coefficient, and doing this for all orders one at a time, is an optimal procedure for deriving the prediction error filter. The minimum value of J_m also has a simple recursive expression, which is explored in Problem 11.10.

Now consider an adaptive solution for the reflection coefficients. First notice that by substituting (11.49) into (11.50) the complex gradient can be written in the much simpler form

$$\nabla_{\gamma_p^*} J_p = -\mathcal{E}\left\{\varepsilon^b_{p-1}[n-1]\varepsilon^*_p[n] + \varepsilon^b_p[n]\varepsilon^*_{p-1}[n]\right\} \tag{11.51}$$

A *steepest descent* algorithm can thus be proposed to find the reflection coefficient γ_p that minimizes J_p. This algorithm has the form

$$\begin{aligned}\gamma_p[n+1] &= \gamma_p[n] - \mu \nabla_{\gamma_p^*} J_p \\ &= \gamma_p[n] + \mu\mathcal{E}\left\{\varepsilon^b_{p-1}[n-1]\varepsilon^*_p[n] + \varepsilon^b_p[n]\varepsilon^*_{p-1}[n]\right\}\end{aligned}$$

Notice that since J_p is a quadratic function of γ_p, this algorithm will converge for sufficiently small values of the step size parameter μ. Furthermore, we can define an LMS-type algorithm by dropping the expectation and taking our step in the direction of the instantaneous gradient. The resulting Gradient Adaptive Lattice (GAL) algorithm uses a time-varying step size parameter analogous to that in the normalized LMS algorithm (11.42) and is given by

$$\boxed{\gamma_p[n+1] = \gamma_p[n] + \frac{\mu'}{\mathcal{E}_{p-1}[n]}\left[\varepsilon^b_{p-1}[n-1]\varepsilon^*_p[n] + \varepsilon^b_p[n]\varepsilon^*_{p-1}[n]\right] \qquad (11.52)}$$

where μ' is a constant and $\mathcal{E}_p$ is a filtered estimate of the total (forward plus backward) error variance satisfying

$$\boxed{\mathcal{E}_p[n] = \beta\mathcal{E}_p[n-1] + (1-\beta)\left[|\varepsilon_p[n]|^2 + |\varepsilon^b_p[n-1]|^2\right]} \tag{11.53}$$

$(0 < \beta < 1)$. As we have seen before, this type of filtered estimate is *asymptotically unbiased*. The step size parameter μ' is theoretically confined to satisfy $0 < \mu' < 2$ but is usually taken as $\mu' < 0.1$. The complete algorithm is obtained by applying (11.49), (11.53) and (11.52) and is summarized below:

GAL Algorithm

1. (Initialization) For $p = 1, 2, \ldots, P$ set $\varepsilon_p[-1] = \varepsilon_p^b[-1] = 0$, $\gamma_p[0] = 0$, and $\mathcal{E}_p[-1] = \epsilon$ (a small number), for $p = 0, 1, 2, \ldots, P$.

2. For $n = 0, 1, 2, \ldots$

   (a) Set $\varepsilon_0[n] = \varepsilon_0^b[n] = x[n]$.
   Compute $\mathcal{E}_0[n] = \beta\mathcal{E}_0[n-1] + (1-\beta)\left[|\varepsilon_0[n]|^2 + |\varepsilon_0^b[n-1]|^2\right]$

   (b) For $p = 1, 2, \ldots, P$ compute

$$\begin{aligned}
\varepsilon_p[n] &= \varepsilon_{p-1}[n] - \gamma_p^*[n]\,\varepsilon_{p-1}^b[n-1] \\
\varepsilon_p^b[n] &= \varepsilon_{p-1}^b[n-1] - \gamma_p[n]\,\varepsilon_{p-1}[n] \\
\mathcal{E}_p[n] &= \beta\mathcal{E}_p[n-1] + (1-\beta)\left[|\varepsilon_p[n]|^2 + |\varepsilon_p^b[n-1]|^2\right] \\
\gamma_p[n+1] &= \gamma_p[n] + \frac{\mu'}{\mathcal{E}_{p-1}[n]}\left[\varepsilon_{p-1}^b[n-1]\varepsilon_p^*[n] + \varepsilon_p^b[n]\varepsilon_{p-1}^*[n]\right]
\end{aligned}$$

A section of MATLAB code to implement the algorithm is given below. Because of the dual recursion (order as well as time) the implementation in MATLAB cannot easily take advantage of array operations and thus is not particularly efficient. The algorithm itself is computationally efficient, however and this efficiency can be realized when coding in languages closer to the machine level.

MAIN PART OF FUNCTION
(GAL ALGORITHM):

```
ef(:,1)=x; eb(:,1)=x;                                    % initialization
x2=x.*conj(x);

for n=2:L+1
   Efb(n,1)=beta*Efb(n-1,1) + beta1*(x2(n) + x2(n-1));
   for p=2:P+1
       ef(n,p)=ef(n,p-1)-conj(Gm(n,p))*eb(n-1,p-1);
       eb(n,p)=eb(n-1,p-1)-Gm(n,p)*ef(n,p-1);
       Efb(n,p)=beta*Efb(n-1,p) ...
            + beta1*(ef(n,p)*conj(ef(n,p)) + eb(n-1,p)*conj(eb(n-1,p)));
       Gm(n+1,p)=Gm(n,p) + (mu1/Efb(n,p-1))...
            *(eb(n-1,p-1)*conj(ef(n,p)) +  eb(n,p)*conj(ef(n,p-1)));
   end
end
```

11.4.2 Joint Process Estimation

It was shown in Chapter 8, Section 8.12 that the Wiener filter for a general estimation problem could be implemented in lattice form, using the basic linear prediction lattice structure as the backbone. The structure of the filter is referred to as a lattice-ladder structure (see Fig. 11.11) and this formulation, commonly referred to as "joint process estimation," is especially useful for an adaptive filter. Note that for an FIR Wiener filter of order P we need a lattice only of order $P-1$. The vector $\tilde{\boldsymbol{x}}[n]$ of P observations is replaced by the vector of backward prediction errors

$$\tilde{\boldsymbol{\varepsilon}}^b[n] = \begin{bmatrix} \varepsilon_0^b[n] \\ \varepsilon_1^b[n] \\ \vdots \\ \varepsilon_{P-1}^b[n] \end{bmatrix} \tag{11.54}$$

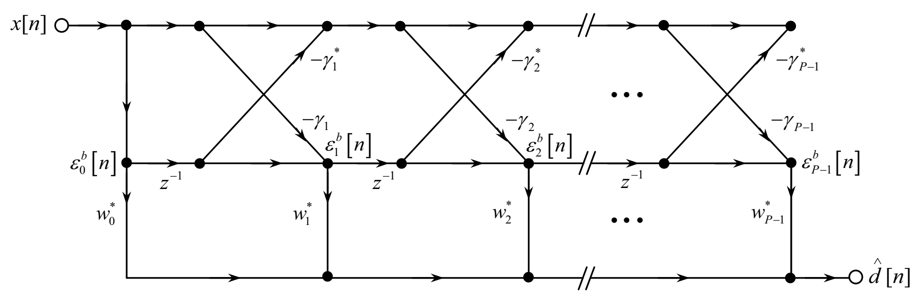


Figure 11.11: Lattice-ladder structure used in joint process estimation.

The vector $\mathbf{w}$ of coefficients w_p for $p = 0, 1, \ldots, P{-}1$ can then be updated adaptively through the LMS algorithm:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu\varepsilon^*[n]\tilde{\boldsymbol{\varepsilon}}^b[n]$$

where $\varepsilon[n] = d[n] - \hat{d}[n]$. Note that in this form of the algorithm the last term on the right hand side of the equation is conjugated with respect to the corresponding term in (11.31). This is because the weights in the joint process implementation are defined conjugated (see Fig. 11.11).

Since the input sequence is transformed to a set of orthogonal random variables, convergence will be rapid provided that these variables are properly normalized.[7] A normalized algorithm for the individual weights can thus be specified as

$$w_p[n+1] = w_p[n] + \frac{\mu'}{\sigma_p^2}\,\varepsilon[n]\varepsilon_p^{b*}[n] \qquad p = 0, 1, \ldots, P-1$$

To obtain a completely adaptive algorithm for the joint process configuration, the reflection coefficients of the lattice and the weights w_p can be adapted simultaneously. In this case, the prediction error variances in the last equation are unknown and have to be estimated. A reasonable estimate however is one half of the filtered sum of prediction errors $\mathcal{E}_p[n]/2$. Thus the LMS update equation for the weights becomes

[7]Note that the corrrelation matrix for the vector of backward prediction errors (11.54) is diagonal with elements σ_p^2. Thus the eigenvalue spread is the ratio of prediction error variances σ_0^2/σ_P^2, which can be quite large.

$$\boxed{w_p[n+1] = w_p[n] + \frac{2\mu'}{\mathcal{E}_p[n]}\,\varepsilon[n]\varepsilon_p^{b*}[n] \qquad p = 0, 1, \ldots, P-1} \tag{11.55}$$

where

$$\varepsilon[n] = d[n] - \hat{d}[n] = d[n] - \sum_{p=0}^{P-1} w_p^*[n]\varepsilon_p^b[n] \tag{11.56}$$

In programming, these two equations are added in step 2 of the GAL algorithm. We refer to this joint process algorithm as the Gradient Adaptive Lattice-Ladder (GAL^2) algorithm. A portion of the MATLAB code for the GAL^2 algorithm is given below. Note how the additional steps are added at the end of the outer loop.

MAIN PART OF FUNCTION
(GAL^2 Algorithm):

```
ef(:,1)=x; eb(:,1)=x;                                    % initialization
x2=x.*conj(x);

for n=2:L+1
   Efb(n,1)=beta*Efb(n-1,1) + beta1*(x2(n) + x2(n-1));
   for p=2:P
       ef(n,p)=ef(n,p-1)-conj(Gm(n,p))*eb(n-1,p-1);
       eb(n,p)=eb(n-1,p-1)-Gm(n,p)*ef(n,p-1);
       Efb(n,p)=beta*Efb(n-1,p) ...
            + beta1*(ef(n,p)*conj(ef(n,p)) + eb(n-1,p)*conj(eb(n-1,p)));
       Gm(n+1,p)=Gm(n,p) + (mu1/Efb(n,p-1))...
            *(eb(n-1,p-1)*conj(ef(n,p)) +  eb(n,p)*conj(ef(n,p-1)));
   end

   err(n) = d(n) - eb(n,:)*W(n,:)';
   W(:,n+1) = W(:,n) + 2*mu1*err(n)*eb(:,n)'./Efb(:,n));
end;
```

11.5 THE RLS ALGORITHM

In Chapter 9 we introduced the methods of least squares. Recall that the methods based on least squares principles are *data oriented*, i.e., the measures of performance are based on errors computed from whatever data is at hand. There is no mention of expectation or ensemble averages. The Recursive Least Squares (RLS) method is an *adaptive* method based on least squares. As such, it is fundamentally different from the adaptive methods presented thus far in this chapter. From a practical viewpoint, it provides another alternative for dealing with adaptive filtering problems. It will be seen that among other things, this method provides for faster convergence at the price of higher computation. (The basic method requires $\mathcal{O}(P^2)$ operations rather than $\mathcal{O}(P)$.) The filtering equations are more extensive however, and can be compared to those of the Kalman filter.

This section reviews the problem of least squares and develops the basic RLS algorithm. Subsequent sections in this chapter discuss a lattice oriented version of the algorithm and even a "fast" RLS algorithm that requires only $\mathcal{O}(P)$ computations.

[Note: Since this discussion makes frequent references to Chapter 9 of the text, you will want to have a copy of that chapter in front of you.]

11.5.1 Weighted Least Squares

Recall the general filtering problem illustrated in Fig. 9.4 on page 520 of the text. A desired data sequence denoted by $\mathrm{d}[n]$ is to be estimated from a related observed data sequence $\mathrm{x}[n]$ by a linear FIR filter. In keeping with the common notation for adaptive filtering, let us denote the filter weights by $w_0, \ldots, w_P$ and define the weight vector $\mathbf{w}$ as in (11.3). The optimal filter that minimizes the sum of squared errors over the obsevation interval $n_I \leq n \leq n_F$ is then given by the solution to the least squares Wiener-Hopf equation

$$(\mathbf{X}^{*T}\mathbf{X})\mathbf{w} = \mathbf{X}^{*T}\mathbf{d}$$

where $\mathbf{X}$ is the data matrix given by (9.45) and $\mathbf{d} = \begin{bmatrix} \mathrm{d}[n_I] & \mathrm{d}[n_I+1] & \cdots & \mathrm{d}[n_F] \end{bmatrix}^T$ is the vector of desired data given in the problem. This equation can be solved by any number of methods discussed in Chapter 9.

In preparation for the adaptive least squares problem, let us consider a generalization of the previous least squares problem and the criterion (9.42). In particular, let the error $\epsilon[n]$ be defined as in (9.41) and define the *weighted* sum of squared errors as

$$\mathcal{S}_\beta[n] \overset{\text{def}}{=} \sum_{i=n_I}^{n} \beta^{n-i}|\epsilon[i]|^2\,; \qquad 0<\beta<1 \tag{11.57}$$

Here the final time n_F has been replaced by a general time n ($n \geq n_I$) and the weighted sum has been defined to show its explicit dependence on n. Since β is a positive number less than 1, this sum gives the highest weight to the error at the current time n and exponentially less weight to errors as they go back in time. As a result, β is sometimes referred to as an *exponential* weighting factor or a "forgetting factor." The filter weight vector $\mathbf{w}[n]$ to minimize this weighted sum of errors (11.57) can be found as follows.

Let $\mathbf{X}[n]$, $\mathbf{d}[n]$ and $\hat{\mathbf{d}}_n$ all denote the corresponding variables at a general time $n_F = n$. In particular

$$\mathbf{X}[n] = \begin{bmatrix} \mathrm{x}[n_I] & \mathrm{x}[n_I-1] & \cdots & \mathrm{x}[n_I-P+1] \\ \mathrm{x}[n_I+1] & \mathrm{x}[n_I] & \cdots & \mathrm{x}[n_I-P+2] \\ \vdots & \vdots & & \vdots \\ \\ \vdots & \vdots & & \vdots \\ \mathrm{x}[n] & \mathrm{x}[n-1] & \cdots & \mathrm{x}[n-P+1] \end{bmatrix} \tag{11.58}$$

and

$$\mathbf{d}[n] = \begin{bmatrix} \mathrm{d}[n_I] \\ \mathrm{d}[n_I+1] \\ \vdots \\ \\ \vdots \\ \mathrm{d}[n] \end{bmatrix} \tag{11.59}$$

Following the approach of Chapter 9, define the error vector at time n as

$$\boldsymbol{\epsilon}[n] = \mathbf{d}[n] - \hat{\mathbf{d}}[n]$$

where the estimated data vector can be written as

$$\hat{\mathbf{d}}[n] = \mathbf{X}[n]\mathbf{w}[n]$$

(see (9.43) and the equation preceding it). Thus

$$\boldsymbol{\epsilon}[n] = \mathbf{d}[n] - \mathbf{X}[n]\mathbf{w}[n] \tag{11.60}$$

Then (11.57) can be written in matrix notation as

$$\mathcal{S}_\beta[n] = \boldsymbol{\epsilon}[n]^{*T}\mathbf{B}[n]\boldsymbol{\epsilon}[n] \tag{11.61}$$

where $\mathbf{B}[n]$ is the diagonal matrix

$$\mathbf{B}[n] = \begin{bmatrix} \beta^{n-n_I} & 0 & \cdots & 0 \\ 0 & \ddots & & \\ \vdots & & \beta & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \tag{11.62}$$

Now by substituting (11.60) once on the left, (11.61) can be rewritten as

$$\mathcal{S}_\beta[n] = (\mathbf{d}[n] - \mathbf{X}[n]\mathbf{w}[n])^{*T}\mathbf{B}[n]\boldsymbol{\epsilon}[n] = \mathbf{d}[n]^{*T}\mathbf{B}[n]\boldsymbol{\epsilon}[n] - \mathbf{w}^{*T}\mathbf{X}^{*T}[n]\mathbf{B}[n]\boldsymbol{\epsilon}[n] \tag{11.63}$$

A necessary condition for minimizing $\mathcal{S}_\beta[n]$ is thus given by

$$\nabla_{\mathbf{w}^*}\mathcal{S}_\beta[n] = -\mathbf{X}^{*T}[n]\mathbf{B}[n]\boldsymbol{\epsilon}[n] = \mathbf{0}$$

(We can safely write this since $\boldsymbol{\epsilon}[n]$ is a function of $\mathbf{w}$, not $\mathbf{w}^*$.) The condition

$$\boxed{\mathbf{X}^{*T}[n]\mathbf{B}[n]\boldsymbol{\epsilon}[n] = \mathbf{0}} \tag{11.64}$$

is actually a statement of the *Least Squares Orthogonality Principle* using a weighted form of inner product. (Compare (11.64) with the first equation in Theorem 9.1, page 524.) Now substituting (11.60) into (11.64) yields

$$\mathbf{X}^{*T}[n]\mathbf{B}[n](\mathbf{d}[n] - \mathbf{X}[n]\mathbf{w}[n]) = \mathbf{0}$$

and leads to the generalized least squares Wiener-Hopf equation

$$\boxed{\mathbf{R}[n]\mathbf{w}[n] = \mathbf{r}[n]} \tag{11.65}$$

where $\mathbf{R}[n]$ and $\mathbf{r}[n]$ are defined by

$$\mathbf{R}[n] = \mathbf{X}^{*T}[n]\mathbf{B}[n]\mathbf{X}[n] \tag{11.66}$$

and

$$\mathbf{r}[n] = \mathbf{X}^{*T}[n]\mathbf{B}[n]\mathbf{d}[n] \tag{11.67}$$

These variables can be thought of as *estimates* of the correlation and cross-correlation of the data, weighted so that older data has less influence.

The solution of (11.65) gives the required filter tap weight vector as

$$\mathbf{w}[n] = \mathbf{R}^{-1}[n]\mathbf{r}[n] \tag{11.68}$$

The minimum weighted sum of squared errors corresponding to the solution of (11.65) can be obtained by observing that when the orthogonality condition (11.64) is satisfied, the last term in (11.63) is actually zero. Thus the weighted sum of squared errors corresponding to the optimal filter is given by [8]

$$\boxed{\mathcal{S}_\beta[n] = \mathbf{d}[n]^{*T}\mathbf{B}[n]\boldsymbol{\epsilon}[n]} \tag{11.69}$$

This corresponds to the *second* equation in Theorem 9.1. By once again substituting (11.60) this can be written in the more explicit form

$$\mathcal{S}_\beta[n] = \mathbf{d}[n]^{*T}\mathbf{B}[n]\mathbf{d}[n] - \mathbf{d}[n]^{*T}\mathbf{B}[n]\mathbf{X}[n]\mathbf{w}[n]$$

or, by taking note of (11.67) and (11.68),

$$\begin{aligned} \mathcal{S}_\beta[n] &= \mathbf{d}[n]^{*T}\mathbf{B}[n]\mathbf{d}[n] - \mathbf{r}^{*T}[n]\mathbf{w}[n] \\ &= \mathbf{d}[n]^{*T}\mathbf{B}[n]\mathbf{d}[n] - \mathbf{r}^{*T}[n]\mathbf{R}^{-1}[n]\mathbf{r}[n] \end{aligned} \tag{11.70}$$

11.5.2 Recursive Least Squares

Having developed a more general form of least squares filtering problem and the generalized Wiener-Hopf equation (11.65), let us now consider finding a recursive solution to this problem. The motivation for doing this is as follows. The solution to (11.65) is for data in the interval $[n_I, n]$. If the solution to this problem can be updated as more data $\mathrm{x}[n+1], \mathrm{x}[n+2]$ etc. is observed, while at the same time older data is deemphasized due to the parameter β in (11.57), then the filter becomes adaptive to the new data.

Our development of the basic RLS algorithm may seem a bit lengthy; however, many important ideas are developed along the way that are useful

[8] To be precise, we should use separate notation for this minimum sum of errors corresponding to the optimal filter to distinguish it from the general expression (11.63) that applies to *any* filter. The distinction however should be clear from the context.

to the understanding of the algorithm and its extension, for example to fast algorithms. The following discussion is therefore divided into subsections, each of which develops one key idea. The results are then brought together at the end to state the most common form of the algorithm.

Recursion for the filter coefficients.

A logical place to begin is to try to express the variables in the least squares Wiener-Hopf equation (11.65) in a recursive form. First note by from (11.58) that the data matrix $\mathbf{X}[n]$ can be written as

$$\mathbf{X}[n] = \begin{bmatrix} \mathbf{X}[n-1] \\ \tilde{\mathbf{x}}^T[n] \end{bmatrix} \tag{11.71}$$

where $\tilde{\mathbf{x}}^T[n]$ represents the last row of the matrix. In other words, $\mathbf{x}[n]$ is a vector defined by

$$\mathbf{x}[n] = \begin{bmatrix} \mathrm{x}[n-P+1] \\ \vdots \\ \mathrm{x}[n-1] \\ \mathrm{x}[n] \end{bmatrix} \tag{11.72}$$

and $\tilde{\mathbf{x}}^T[n]$ is its reversed transposed form. Also note from (11.59) and (11.62) that $\mathbf{d}[n]$ and $\mathbf{B}[n]$ can be written recursively as

$$\mathbf{d}[n] = \begin{bmatrix} \mathbf{d}[n-1] \\ \mathrm{d}[n] \end{bmatrix} \tag{11.73}$$

and

$$\mathbf{B}[n] = \begin{bmatrix} \beta\mathbf{B}[n-1] & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tag{11.74}$$

By applying these results to (11.66) and (11.67), the terms in the Wiener-Hopf equation can be written recursively as

$$\begin{aligned} \mathbf{R}[n] &= \mathbf{X}^{*T}[n]\mathbf{B}[n]\mathbf{X}[n] \\ &= \begin{bmatrix} \mathbf{X}^{*T}[n-1] & \tilde{\mathbf{x}}^*[n] \end{bmatrix} \begin{bmatrix} \beta\mathbf{B}[n-1] & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}[n-1] \\ \tilde{\mathbf{x}}^T[n] \end{bmatrix} \\ &= \beta\mathbf{R}[n-1] + \tilde{\mathbf{x}}^*[n]\tilde{\mathbf{x}}^T[n] \end{aligned} \tag{11.75}$$

and

$$\begin{aligned}\mathbf{r}[n] &= \mathbf{X}^{*T}[n]\mathbf{B}[n]\mathbf{d}[n] \\ &= \begin{bmatrix} \mathbf{X}^{*T}[n-1] & \tilde{\mathbf{x}}^{*}[n] \end{bmatrix} \begin{bmatrix} \beta\mathbf{B}[n-1] & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{d}[n-1] \\ \mathrm{d}[n] \end{bmatrix} \\ &= \beta\mathbf{r}[n-1] + \tilde{\mathbf{x}}^{*}[n]\mathrm{d}[n] \qquad (11.76)\end{aligned}$$

Now, let us write $\mathbf{w}[n]$ as

$$\mathbf{w}[n] = \mathbf{w}[n-1] + \Delta\mathbf{w} \qquad (11.77)$$

and attempt to find a convenient form for $\Delta\mathbf{w}$. Substituting this expression in (11.65) and expanding yields

$$\mathbf{R}[n]\mathbf{w}[n-1] + \mathbf{R}[n]\Delta\mathbf{w} = \mathbf{r}[n]$$

Further substitution of (11.75) and (11.76) for the first and last terms in this equation leads to

$$\left(\underbrace{\beta\mathbf{R}[n-1]} + \tilde{\mathbf{x}}^{*}[n]\tilde{\mathbf{x}}^{T}[n]\right)\mathbf{w}[n-1] + \mathbf{R}[n]\Delta\mathbf{w} = \underbrace{\beta\mathbf{r}[n-1]} + \tilde{\mathbf{x}}^{*}[n]\mathrm{d}[n]$$

Now notice, that using (11.65) once again, the terms with the underbraces can be dropped out of the equation. Thus, by rearranging terms, the last equation can be rewritten as

$$\begin{aligned}\mathbf{R}[n]\Delta\mathbf{w} &= \tilde{\mathbf{x}}^{*}[n]\left(\mathrm{d}[n] - \tilde{\mathbf{x}}^{T}[n]\mathbf{w}[n-1]\right) \\ &= \tilde{\mathbf{x}}^{*}[n]\,\mathrm{e}[n] \qquad (11.78)\end{aligned}$$

The new variable $\mathrm{e}[n]$ is known as the the *a priori* error or the *prior* error; it has been defined implicitely above and can be written as

$$\boxed{\mathrm{e}[n] = \mathrm{d}[n] - \mathbf{w}^{T}[n-1]\tilde{\mathbf{x}}[n]} \qquad (11.79)$$

The prior error is the error resulting from applying the filter coeffiecients for time $n-1$ to the data at time n. It is *not* the same as the error $\epsilon[n]$ that appears in the weighted sum (11.57). That error is given by

$$\boxed{\epsilon[n] = \mathrm{d}[n] - \mathbf{w}^{T}[n]\tilde{\mathbf{x}}[n]} \qquad (11.80)$$

and is sometimes referred to as the *a posteriori* error or the *posterior* error. Both $\mathrm{e}[n]$ and $\epsilon[n]$ are important for the RLS algorithm equations to be developed.

Now, let us return to (11.77) and introduce a vector $\mathbf{k}[n]$ such that

$$\Delta\mathbf{w} = \mathbf{k}[n]\mathrm{e}[n]$$

This form for $\Delta\mathbf{w}$ seems reasonable, since you could expect that the change in weight vector should be proportional to the error. With this formulation the weight update equation (11.77) becomes

$$\boxed{\mathbf{w}[n] = \mathbf{w}[n-1] + \mathbf{k}[n]\mathrm{e}[n]} \tag{11.81}$$

and it follows from (11.78) that $\mathbf{k}[n]$ is the solution to

$$\boxed{\mathbf{R}[n]\mathbf{k}[n] = \tilde{\mathbf{x}}^*[n]} \tag{11.82}$$

It may seem that in deriving these last two equations, we have simply traded one problem for another. In particular, by comparing (11.82) and (11.65) it can be seen that $\mathbf{k}[n]$ is just the solution to the Wiener-Hopf equation with $\mathbf{r}[n]$ replaced by $\tilde{\mathbf{x}}^*[n]$. Equation 11.81 is a desirable form for the adaptive filter however, because it shows how the error (the prior error in this case) is used to update the filter coefficients. The term $\mathbf{k}[n]$ is called the "gain" vector in analogy to a similar term appearing in the Kalman filtering equations. Its significance as a solution to (11.82) is revealed later in the chapter.

Relation between prior and posterior errors.

The prior and posterior errors can be related using (11.80), (11.81) and (11.79) as follows:

$$\begin{aligned}
\epsilon[n] &= \mathrm{d}[n] - \mathbf{w}^T[n]\tilde{\mathbf{x}}[n] \\
&= \mathrm{d}[n] - (\mathbf{w}[n-1] + \mathbf{k}[n]\mathrm{e}[n])^T\,\tilde{\mathbf{x}}[n] \\
&= (\mathrm{d}[n] - \mathbf{w}^T[n-1]\tilde{\mathbf{x}}[n]) - \mathrm{e}[n]\mathbf{k}^T[n]\tilde{\mathbf{x}}[n] \\
&= \mathrm{e}[n] - \mathrm{e}[n]\mathbf{k}^T[n]\tilde{\mathbf{x}}[n]
\end{aligned}$$

Then factoring out $\mathrm{e}[n]$ simplifies this equation to

$$\boxed{\epsilon[n] = \kappa[n]\mathrm{e}[n]} \tag{11.83}$$

where

$$\boxed{\kappa[n] = 1 - \mathbf{k}^T[n]\tilde{\mathbf{x}}[n]} \tag{11.84}$$

is called the *conversion factor*. This variable is further explored in Problem 11.12 where it is shown that $\kappa[n]$ is *real* and satisfies

$$0 < \kappa[n] < 1 \tag{11.85}$$

Note that by using the relation (11.83), the update equation (11.81) for the filter weights can be written in the alternative form

$$\mathbf{w}[n] = \mathbf{w}[n-1] + \mathbf{k}'[n]\epsilon[n] \tag{11.86}$$

involving the posterior error, where the new gain term $\mathbf{k}'[n]$ is related to $\mathbf{k}[n]$ by

$$\mathbf{k}[n] = \kappa[n]\,\mathbf{k}'[n] \tag{11.87}$$

You can also easily show that in analogy to (11.82), $\mathbf{k}'[n]$ is the solution to

$$\beta\mathbf{R}[n-1]\mathbf{k}'[n] = \tilde{\mathbf{x}}^*[n] \tag{11.88}$$

and that an alternate expression for the conversion factor is

$$\kappa[n] = \frac{1}{1 + \mathbf{k}'^T[n]\tilde{\mathbf{x}}[n]} \tag{11.89}$$

(see Problem 11.12).

Recursion for the inverse correlation matrix.

In order to have a completely recursive algorithm, it is necessary to have a recursive expression for the gain $\mathbf{k}[n]$ that appears in (11.81). Since $\mathbf{k}[n]$ is the solution to (11.82), let us focus on finding a recursive expression for the inverse correlation matrix $\mathbf{R}^{-1}[n]$. Fortunately such a result is available for matrices of the form (11.75). This result is known to electrical engineers as simply the "matrix inversion lemma." Mathematicians however, refer to it as the Sherman-Morrison-Woodbury formula, with the simplest statement of the result given by Sherman and Morrison and the most general case due to Woodbury. In the simplest (Sherman-Morrison) form the result states that

$$(\mathbf{A} + \mathbf{b}\mathbf{c}^{*T})^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{b}\,\mathbf{c}^{*T}\mathbf{A}^{-1}}{1 + \mathbf{c}^{*T}\mathbf{A}^{-1}\mathbf{b}}$$

where $\mathbf{A}$ is a nonsingular matrix and $\mathbf{b}$ and $\mathbf{c}$ are conformable column vectors.

Applying this formula to (11.75) yields

$$\mathbf{R}^{-1}[n] = (\beta\mathbf{R}[n-1])^{-1} - \frac{(\beta\mathbf{R}[n-1])^{-1}\tilde{\mathbf{x}}^*[n]\tilde{\mathbf{x}}^T[n](\beta\mathbf{R}[n-1])^{-1}}{1+\tilde{\mathbf{x}}^T[n](\beta\mathbf{R}[n-1])^{-1}\tilde{\mathbf{x}}^*[n]}$$

Now, examine this expression in light of previous results. Using (11.88) and (11.89), the *denominator* of the fraction above can be identified as

$$1+\tilde{\mathbf{x}}^T[n](\beta\mathbf{R}[n-1])^{-1}\tilde{\mathbf{x}}^*[n] = 1+\tilde{\mathbf{x}}^T[n]\mathbf{k}'[n] = 1/\kappa[n]$$

The *numerator* can be written as the product

$$\Big((\beta\mathbf{R}[n-1])^{-1}\tilde{\mathbf{x}}^*[n]\Big)\Big((\beta\mathbf{R}[n-1])^{-1}\tilde{\mathbf{x}}^*[n]\Big)^{*T} = \mathbf{k}'[n]\mathbf{k}'^{*T}[n]$$

(where we used the fact that $\mathbf{R}[n-1]$ is Hermitian symmetric and β is real). Putting the last three equations together, and using (11.87) then produces

$$\boxed{\mathbf{R}^{-1}[n] = \beta^{-1}\mathbf{R}^{-1}[n-1] - \mathbf{k}[n]\mathbf{k}'^{*T}[n]} \qquad (11.90)$$

which is the final desired result.

The complete RLS algorithm is obtained by combining (11.88), (11.89), (11.87), (11.79), (11.81) and (11.90), in that order. The algorithm is summarized in (11.91) below:

$$\boxed{\begin{array}{rcll}
\mathbf{k}'[n] & = & \beta^{-1}\mathbf{R}^{-1}[n-1]\tilde{\mathbf{x}}^*[n] & \text{(a)} \\
\kappa[n] & = & 1/(1+\mathbf{k}'^T[n]\tilde{\mathbf{x}}[n]) & \text{(b)} \\
\mathbf{k}[n] & = & \kappa[n]\mathbf{k}'[n] & \text{(c)} \\
\mathrm{e}[n] & = & \mathrm{d}[n] - \mathbf{w}^T[n-1]\tilde{\mathbf{x}}[n] & \text{(d)} \\
\mathbf{w}[n] & = & \mathbf{w}[n-1] + \mathbf{k}[n]\mathrm{e}[n] & \text{(e)} \\
\mathbf{R}^{-1}[n] & = & \beta^{-1}\mathbf{R}^{-1}[n-1] - \mathbf{k}[n](\mathbf{k}'[n])^{*T} & \text{(f)}
\end{array} \quad \mathsf{RLS} \quad (11.91)}$$

These are sometimes written in textbooks as a smaller set of equations, but the expanded version shown here identifies all the significant variables and moreover eliminates any tendancy to perform redundant computation. Steps (a), (b), and (c) deal with the adaptive gain computation, step (d) is the filtering and error computation, and step (e) is the weight update. Step (f) computes the inverse correlation matrix in preparation for the next pass. These equations can be put in direct correspondence with the Kalman filter equations of Chapter 7.

The recursion can be initialized in one of two ways. First, you can start by formulating and solving the Wiener-Hopf equations for a value $n \geq n_I + P$ to obtain the weight vector and inverse correlation matrix. Thereafter you can apply the recursion to have an exact solution at each step. It is easier, however, to begin with initial conditions

$$\begin{aligned}\mathbf{R}^{-1}[0] &= \frac{1}{\delta}\mathbf{I}\\ \mathbf{w}[0] &= \mathbf{0}\end{aligned}$$

where δ is a small positive constant. This quickly converges to the exact solution as n gets large.

A final topic that needs to be addressed is the weighted sum of squared errors, which can also be computed recursively. This is not as simple as it seems, because in recomputing (11.57) for n instead of $n-1$ there is *a whole new set of error terms* involved.

To develop the recursive formula, start with the general expression (11.69) and consider the form of the error vector $\boldsymbol{\epsilon}$ that appears in that equation. Using (11.60), (11.81), and the partitionings (11.71) and (11.73), we can write

$$\begin{aligned}\boldsymbol{\epsilon}[n] = \mathbf{d}[n] - \mathbf{X}[n]\mathbf{w}[n] &= \mathbf{d}_n - \mathbf{X}[n](\mathbf{w}[n-1] + \mathbf{k}[n]\mathrm{e}[n])\\ &= \begin{bmatrix} \mathbf{d}[n-1] - \mathbf{X}[n-1]\mathbf{w}[n-1] \\ \mathrm{d}[n] - \tilde{\mathbf{x}}^T[n]\mathbf{w}[n-1] \end{bmatrix} - \mathbf{X}[n]\mathbf{k}[n]\mathrm{e}[n]\\ &= \begin{bmatrix} \boldsymbol{\epsilon}[n-1] \\ \mathrm{e}[n] \end{bmatrix} - \mathbf{X}[n]\mathbf{k}[n]\mathrm{e}[n]\end{aligned}$$

where (11.60) and (11.79) were used to write the last step. Substituting this result into (11.69) and using the partitionings (11.73) and (11.74) produces

$$\begin{aligned}
\mathcal{S}_\beta[n] &= \mathbf{d}^{*T}[n]\mathbf{B}[n]\boldsymbol{\epsilon}[n] \\
&= \begin{bmatrix} \mathbf{d}^{*T}[n-1] & \mathrm{d}^*[n] \end{bmatrix} \begin{bmatrix} \beta\mathbf{B}[n-1] & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\epsilon}[n-1] \\ \mathrm{e}[n] \end{bmatrix} - \mathbf{d}^{*T}[n]\mathbf{B}[n]\mathbf{X}[n]\mathbf{k}[n]\mathrm{e}[n] \\
&= \beta\mathcal{S}_\beta[n-1] + \mathrm{d}^*[n]\mathrm{e}[n] - \mathbf{r}^{*T}[n]\mathbf{k}[n]\mathrm{e}[n]
\end{aligned}$$

where (11.69) and (11.67) were used to simplify in the last step. Now, the product $\mathbf{r}^{*T}[n]\mathbf{k}[n]$ appearing in the last line can be rewritten using (11.65) and (11.82) as

$$\mathbf{r}^{*T}[n]\mathbf{k}[n] = (\mathbf{R}[n]\mathbf{w}[n])^{*T}\mathbf{k}[n] = \mathbf{w}^{*T}[n]\mathbf{R}[n]\mathbf{k}[n] = \mathbf{w}^{*T}[n]\tilde{\mathbf{x}}^*[n]$$

Thus the previous equation becomes

$$\mathcal{S}_\beta[n] = \beta\mathcal{S}_\beta[n-1] + \left(\mathrm{d}^*[n] - \mathbf{w}^{*T}[n]\tilde{\mathbf{x}}^*[n]\right)\mathrm{e}[n]$$

or, in view of (11.80),

$$\boxed{\mathcal{S}_\beta[n] = \beta\mathcal{S}_\beta[n-1] + \epsilon^*[n]\mathrm{e}[n]} \tag{11.92}$$

which is the desired result. By using the conversion relation (11.83) this last equation can be written in two other alternate forms:

$$\boxed{\begin{aligned}
\mathcal{S}_\beta[n] &= \beta\mathcal{S}_\beta[n-1] + \kappa[n]|\mathrm{e}[n]|^2 && \text{(a)} \\
\mathcal{S}_\beta[n] &= \beta\mathcal{S}_\beta[n-1] + |\epsilon[n]|^2/\kappa[n] && \text{(b)}
\end{aligned}} \tag{11.93}$$

The main loop for a MATAB implementation of the RLS algorithm is shown below:

MAIN LOOP OF FUNCTION
(RLS ALGORITHM):

```
for n=P:length(x) % loop through all data
xn=x(n:-1:n-P+1);
g=Rm1*xn';
gamma=1/(beta + xn*g);
k=gamma*g;
epr(n) = d(n) - xn*W(:,n-1);
W(:,n) = W(:,n-1) + k*epr(n);
Rm1 = (Rm1 - k*g')/beta;
epo(n) = beta*gamma*epr(n);
S(n) = beta*S(n-1) + epo(n)*conj(epr(n));
end
```

The data vector 'x' appearing in the second line is assumed to be a row vector, hence 'xn' is also a row vector. Note that in this implementation equations (11.91) have been modified slightly to achieve more efficient computation (see Problem 11.14). In particular, two new variables have been defined as

$$\mathbf{g}[n] = \beta\mathbf{k}'[n] \qquad\qquad \gamma[n] = \kappa[n]/\beta \tag{11.94}$$

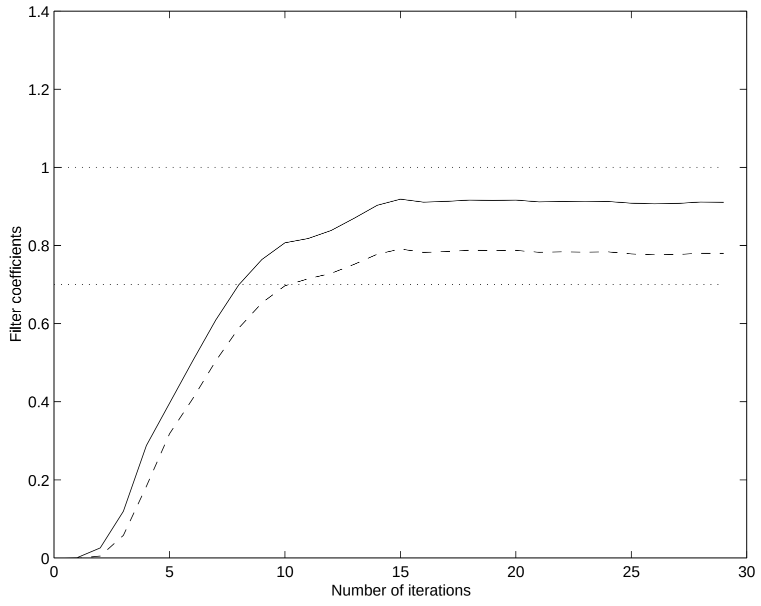
and the computations are carried out using these terms. The variable 'W' is a matrix whose columns represent the filter coeffiecent vector as a function of time, and 'epr' and 'epo' represent the prior and posterior errors respectively.

11.5.3 RLS Performance

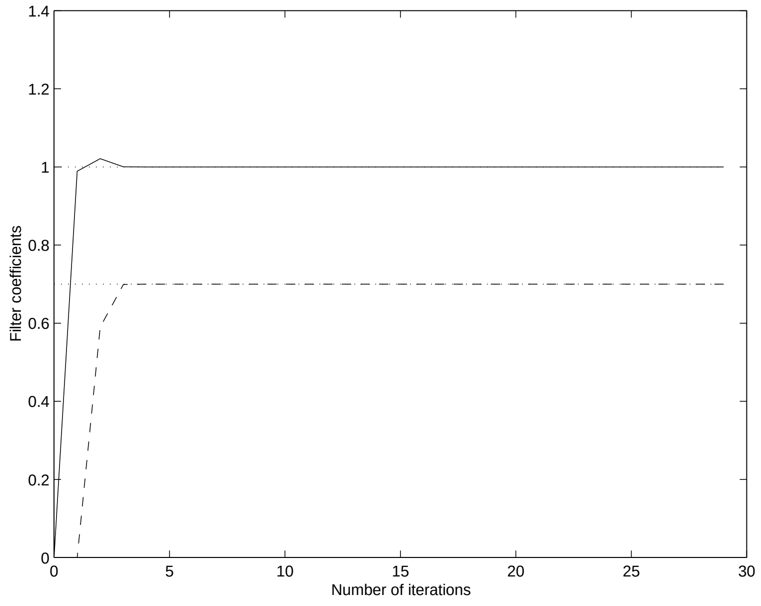
Example 11.2 The RLS method is applied to the system identification problem considered in Example 11.1. Recall that when the input process is highly correlated convergence is very slow (see Fig. 11.8 (d)). Figure 11.12 compares the performance of LMS and RLS during the first 30 iterations. From this limited observation time it appears that the values of the filter coefficients produced by the LMS algorithm are leveling off but are nowhere near the true values. (Actually we know from the previous example that the filter coefficients have *not* leveled off and do approach the true values, but only after more than 500 iterations.) The

RLS algorithm however converges to the correct values after just *three* iterations and remains stable. This remarkably better performance is obtained, of course, with a significant increase in the number of computations per iteration.

$\square$



(a)



(b)

Figure 11.12: System identification example. (a) LMS method $a = 0.95, \chi = 39, \mu = 0.01$. (b) RLS method with same input sequence.

Bibliography

[1] B. Widrow et al. Adaptive noise cancelling: Principles and applications. *Proceedings of the IEEE*, 63:1962–1716, 1975.

[2] John J. Shynk. Adaptive IIR filtering. *IEEE ASSP Magazine*, 6(2):4–21, April 1989.

[3] Monson H. Hayes. *Statistical Digital Signal Processing and Modeling.* John Wiley & Sons, New York, 1996.

[4] Simon Haykin. *Adaptive Filter Theory.* Prentice Hall, Inc., Englewood Cliffs, New Jersey, third edition, 1996.

[5] V. John Mathews. Adaptive polynomial filters. *IEEE Signal Processing Magazine*, 8(3):10–26, July 1991.

[6] W. Kenneth Jenkins et al. *Advanced Concepts in Adaptive Signal Processing.* Kluwer, Boston, 1996.

[7] B. Widrow and M. E. Hoff, Jr. Adaptive switching circuits. In *IRE WESCON Convention Record - Part 4*, pages 94–104. Institute of Radio Engineers, 1960.

[8] Odile Macchi. *Adaptive Processing: The Least Mean Squares Approach with Applications in Transmission.* John Wiley & Sons, New York, 1995.

[9] Daniel F. Marshall and W. Kenneth Jenkins. A fast quasi-Newton adaptive filtering algorithm. *IEEE Transactions on Signal Processing*, 40(7):1652–1662, July 1992.

[10] L. J. Griffiths. A continuously adaptive filter implemented as a lattice structure. In *Proceedings of the IEEE International Conference on*

Acoustics, Speech, and Signal Processing, pages 683–686. IEEE Signal Processing Society, 1977. (Hartford).

[11] L. J. Griffiths. An adaptive lattice structure for noise-canceling applications. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 87–90. IEEE Signal Processing Society, 1978. (Tulsa).

Problems for Chapter 11

11.1 Suppose the input $x[n]$ to a second order adaptive filter is a first order random process with parameters a and σ_w^2 as specified in Example 11.1. **11-B**

(a) Show that the eigenvalues of the input correlation matrix are given by

$$\lambda_1 = \frac{\sigma_w^2}{1-|a|} \quad \text{and} \quad \lambda_2 = \frac{\sigma_w^2}{1+|a|}$$

and therefore that the eigenvalue spread is given by

$$\chi = \frac{\lambda_1}{\lambda_2} = \frac{1+|a|}{1-|a|}$$

Use this result to check the values in the table in Example 11.1.

(b) Derive an expression for the misadjustment $\mathcal{M}$ in terms of a and σ_w^2.

11.2 (Problem on convergence of mean using independence theory.) **11-G**

11.3 (Problem on mean-square convergence using independence theory and derivation of expression for $\mathcal{M}$.) **11-H**

11.4 You are given the following data sequencees: **11-M**

$$\begin{aligned} x[n] &= \{\ 1,\ -2,\ 3,\ -4,\ 5\ \} \\ d[n] &= \{\ 1,\ -1,\ 1,\ -1,\ 1\ \} \end{aligned}$$

It is desired to find a filter to estimate $d[n]$ from $x[n]$. The filter is to be of the form:

$$\hat{d}[n] = w_0 x[n] + w_1 x[n-1]$$

(a) Find the weights, w_i, for the optimal (Wiener) filter assuming *no pre-windowing or post-windowing* of the data. Apply the filter to the data and generate the error sequence $e[n] = d[n] - \hat{d}[n]$.

(b) What is the "correlation matrix" used in the solution of this least squares problem? Be sure to use the appropriate normalization.

(c) What is the theoretical upper bound on the step size parameter μ that would allow an LMS algorithm applied to this problem to converge?

(d) What is the condition number of the correlation matrix? Would you expect the LMS algorithm for this problem to converge relatively slowly or relatively rapidly?

11-N 11.5 Apply the LMS algorithm to compute the filter weight vector $\mathbf{w} = [w_0 \;\; w_1]^T$ in Prob. 11.4. Assume there is *no prewindowing* of the data and take $\mu = 0.05$. Start with $\mathbf{w}[0] = \mathbf{0}$ and carry out the algorithm numerically for three steps, i.e., up to $\mathbf{w}[3]$ and $e[3]$.

11-C 11.6 Show that the leaky LMS algorithm can be derived by minimization of the quantity

$$|\varepsilon[n]|^2 + \alpha \|\mathbf{w}[n]\|^2$$

11-D 11.7 Show that the optimum value of $\boldsymbol{\delta}$ for the Normalized LMS algorithm is given by

$$\boldsymbol{\delta}[n+1] = \frac{\varepsilon[n]\tilde{\boldsymbol{x}}^*[n]}{\|\tilde{\boldsymbol{x}}[n]\|^2}$$

11-L 11.8 By taking the expected value of (11.45) and using the formula for the sum of a geometric series, derive a closed-form expression for $\mathcal{E}\left\{\hat{P}_T[n]\right\}$. Use this to show that the limit $P_T = \lim_{n\to\infty} \mathcal{E}\left\{\hat{P}_T[n]\right\}$ exists and is given by $P_T = PR_x[0]$. Therefore $\hat{P}_T[n]$ is an asymptotically unbiased estimate of the mean tap input power.

11-E 11.9 Show that the sign-error algorithm given by (11.46) can be derived by minimizing the absolute value of the instantaneous error $|\varepsilon[n]|$.

11-F 11.10 Show that the minimum value of the error criterion defined by

(11.48) satisfies the order recursion

$$\begin{aligned}(J_p)_{\min} &= (J_{p-1})_{\min} - 2\gamma_p \mathcal{E}\left\{\varepsilon^{b*}_{p-1}[n-1]\varepsilon_{p-1}[n]\right\} \\ &= (J_{p-1})_{\min} - 2|\gamma_p|^2\sigma^2_{p-1}\end{aligned}$$

11.11 Apply the RLS algorithm to the data in Prob. 11.4 with $\beta = 1$. Assume no prewindowing and use the initial conditions **11-P**

$$\mathbf{R}^{-1}[0] = 1000\,\mathbf{I}$$

$$\mathbf{w}[0] = \mathbf{0}$$

Carry out the computations up to $n = 3$ and compute $\mathbf{w}[n]$ and the *posterior* error. Compare the error sequence to that for the LMS method in Prob. 11.5 above.

11.12 By using the relation (11.82), the recursion (11.75) for the correlation matrix, and the definition (11.84) **11-I**

(a) Show that the gain $\mathbf{k}'[n]$ satisfies (11.88).

(b) Also show that an alternate expression for the conversion factor $\kappa[n]$ is (11.89).

(c) Starting with the results in (a) and (b), show that $\kappa[n]$ is real-valued and satisfies (11.85).

11.13 Show that the inverse correlation matrix $\mathbf{R}^{-1}[n]$ in the RLS method satisfies the recursion **11-J**

$$\mathbf{R}^{-1}[n] = \beta^{-1}\mathbf{K}[n]\mathbf{R}^{-1}[n-1]$$

where $\mathbf{K}[n]$ is the matrix

$$\mathbf{K}[n] = \mathbf{I} - \mathbf{k}[n]\tilde{\mathbf{x}}^T[n]$$

11.14 **(a)** Carefully compute and compare the total number of arithmetic operations *per iteration* necessary for implementing the LMS algorithm (11.31) and the RLS algorithm (11.91). **11-K** Count the additions and the multiplications separately, considering subtraction as addition and division as multiplication. Your results should be expressed as polynomials in P, the order of the filter.

(b) Show that by rewriting (11.91) in terms of the variables $\mathbf{g}[n]$ and $\gamma[n]$ defined by (11.94) as is done in the MATLAB code, you can save P multiplications per iteration.

Computer Assignments for Chapter 11

11.1 An “unknown” linear system is described by the difference equation

$$y[n] = x[n] - 0.7x[n-1]$$

where x is the input and y is the output. It is desired to estimate the parameters of this system by using an adaptive filter in the system identification configuration. The adaptive filter is taken to be a first order FIR filter with unknown weights w_0 and w_1.

Four different signals will be used for the input sequence. These are the data sets S00, S01, S02, and S03. These data sets have been generated using a model of the form

$$x[n] = ax[n-1] + v[n]$$

where v is a unit variance white noise sequence and a is equal to 0, 0.95, 0.7 and -0.95 respectively.

(a) Using the results of Problem 11.1, generate a table similar to that in Example 11.1 for these four input data sequences.

(b) Write a MATLAB function to implement the LMS method, and plot the trajectories of the filter coefficients and the squared error (learning curve) of the adaptive filter for each of the four input sequences. For each input choose μ to be 1/10 of the theoretical upper bound.

(c) For each of the input data sets, estimate $R_x[0]$ and compute the upper bound given by equation 11.32. Repeat the simulations of part (b) choosing μ to be 1/10 of these upper bounds.

(d) Determine by experimentation the largest value of μ for each sequence that will not cause the process to become unstable. Provide a table comparing these values to the upper bounds in parts (a) and (c).

(e) Choose one of the variations of the LMS method discussed in Section 11.3.3 and conduct a set of simulations using the four input sequences and an appropriate value of the step size parameter. Compare the results you obtain to the results for the basic LMS method.

11.2 Refer to Computer Assignment 11.1 and apply the RLS method to this system identification problem using each of the input data sets S00, S01, S02, and S03. Using the function 'rls.m' provided to you, plot and compare the weight trajectories and the squared error sequences to those obtained from the LMS algorithm you wrote in the Computer Assignment 11.1. Plot results for RLS and LMS on the same graph for ease of comparison. For each case, *estimate* and compare the number of additions and multiplications required for convergence.